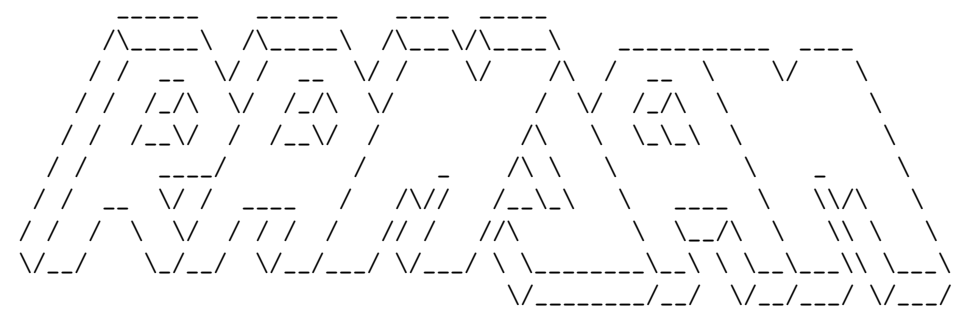
Corso Completo di programmazione assembler in due dischi

Per Amiga 500, 500+, 600 e 1200

di Fabio Ciucci aka Randy/RAMJAM

Proofreading, update dei testi e pagina della versione 2016 del corso
a cura di Alessandro Sperindé

Prefazione, assemblaggio e immagini delle lezioni, sito internet,
testi 2016 a cura di Stefania Calcagno aka Yuki/RamJam



Corso completo di programmazione assembler in due dischi

Versione 1.0.0

INDICE

Elenco delle figure

PREFAZIONE

Fabio *Randy* Ciucci

Inizio a scrivere una prefazione del corso asm dopo 22 anni dalla sua uscita, gli stessi giorni che il Javascript, un linguaggio nemmeno compilato, si è confermato il più popolare del 2015 sia in *StackExchange* che *GitHub*, non solo per web app ma per server con Node e mobile apps.

Nessuno fa caso che ogni progetto tipo *Angular* o altro produce all'incirca 50-100MB di cartella `mode_modules` ad ogni `npm install`, anche per fare un bottone *"hello world"*. La prima volta che provai, esclamai: "Ma stiamo scherzando!?". Non potevo credere che la tecnologia più moderna e in voga del momento, per 4 sorgenti di prova in 4 cartelle, genera 4 copie di librerie quasi uguali, ognuna da 100MB con migliaia di files opensource di dubbia coerenza e utilità, e che solo a me sembrava inefficiente.

L'aver scritto *4kb intros* da piccolo mi fa dimenticare che 400MB sul mio HD SSD turbo da 512GB occupano proporzionalmente meno di una 4kb intro in un dischetto da 700k. E che tutto va in diretta su 8 core a 3Ghz con 16GB RAM e Nvidia spaziale... e sto parlando di un portatile. Pure i telefoni sono così potenti che con *Cordova/Phonegap* gli stessi impasti di *Javascript* e *CSS* sono sempre più usati per fare app Android e iOS, così da non usare Java o Objective C. Tranne che per i giochi più complicati, per quelli se non altro su usa sempre il C++, anche se gli engine 2d e 3d sono scritti da altri, i quali a loro volta chiamano OpenGL e simili che chiamano le schede grafiche. Insomma altre migliaia di files di cui ognuno sa solo il proprio piccolo pezzo pertinente. In asm dai e togli letteralmente la corrente ai circuiti integrati che ti pare, per i millisecondi che ti pare, affinchè arrivi corrente al tale punto nel monitor, al tale motore del disk drive, al tale altoparlante. Non chiami librerie di altri se non all'inizio per dire gentilmente al sistema operativo di togliersi di torno che tanto non serve. Il codice è solo il tuo, salvo se hai rippato, ovvero copia e incollato dei pezzi, ma comunque è tutto e solo nel tuo sorgente, questo controllo assoluto e conoscenza totale fa sentire *sovrannaturali*. Una sensazione che non si sente quando scrivi 100 linee che chiamano un altro milione di linee tra API e sistemi operativi di cui ignori i dettagli e che, diciamocelo, spesso sono piene di bug e le versioni vanno in conflitto tra loro e via dicendo.

D'altronde, il software non ha lo scopo di farti sentire sovrannaturale, ma invece deve risolvere problemi in tempi e costi ridotti, e oggi costa più il tempo del programmatore dello spazio hard disk o velocità CPU/GPU. Infatti, da molti anni scrivo le mie cose in Python quando possibile, perché (a differenza del Javascript) si fa veramente presto, e spesso si rimane compiaciuti della maggiore eleganza e brevità rispetto a C++ o Java, dei cui grovigli di parentesi graffe e cicli "for" si fa certo a meno.

Il più grande cambiamento non è quello che tutti notano ovvero l'hardware e i linguaggi, bensì internet e opensource: ai tempi in cui rilasciai il corso ASM, era l'unica fonte completa di informazioni sull'argomento in italiano. E comunque anche in inglese i libri erano pochi ed evasivi, e soprattutto nessuno rilasciava i codici sorgenti, spesso neppure ai membri dello stesso gruppo. Il codice era una cosa esoterica Elite da tenere segreta sia ai Lamer che alla

concorrenza, esterna e pure interna. L'unica era *decompilare*, una volta per un dramma in Ram Jam che alcuni non davano il sorgente agli altri mi è stato chiesto gentilmente di disassemblare un magazine per poi updatarlo ad AGA, un lavoro che sembra assurdo a pensarci oggi, ma era normale. Fece sia scandalo, perché era un *rippaggio* seppure semi autorizzato, sia ammirazione, perché l'operazione di *hackeraggio* e l'update AGA non erano una *lamerata* come chi copia incolla una scritta scorrevole. Infatti non nascondo che ho anche *crackato* e *trainerato* assai (tanto ormai è in prescrizione), e un decompilato, premesso che di solito non parte, non ha nomi label né commenti né la struttura o formattazione dei moderni opensource a API a cui tutti sono abituati, magari in IDE che mettono in sovraimpressione la documentazione di ogni funzione passandoci sopra col mouse. E comunque qualsiasi problema o domanda la googli, e trovi sempre qualcuno che l'ha risposta. Oggigiorno i programmatori sono degli artisti del copia incolla da StackExchange e simili, io stesso googlo tanto, proprio ricordandomi i tempi in cui non c'era documentazione e l'unica era fare dei loop che provavano tutti i valori in un registro e vedere cosa succedeva, come quando è uscito l'AGA ma la Commodore si è dimenticata di documentarlo.

Tutto è cambiato, ma sempre attuale è il successo di ex demo scener nel passare ai video giochi, essendo le competenze simili e trasferibili. L'esempio supremo, ovviamente in Finlandia, sono i Virtual Dreams (http://demozoo.org/groups/609/), (http://www.pouet.net/groups.php?which=201), la sezione demo dei Fairlight, che vincono l'Assembly party nel 1996 con la demo "Sumea", da cui nel 2001 la società "Sumea" per fare giochi per cellulari (come me negli stessi tempi, con ex scener meno nordici). Poi nel 2010 Supercell (https://en.wikipedia.org/wiki/Supercell_(video_game_company), in 6 persone (ma quasi nessuno dei coder Amiga originari, persi per strada) investono 250mila euro, e fanno i miliardi. Letteralmente, 2 miliardi di euro le vendite del 2015, e 180 dipendenti, grazie a "Clash of Clans" con 100 milioni di giocatori. Anche Rovio, creatore finnico di Angry Birds, brulica di ex scener. In realtà contano molto banche e venture capital in queste cose, per questo in Italia non succede anche ai coder sovrannaturali. In compenso un italiano emigrato in UK (non un coder) ha fondato King, la società di Candy Crush, venduta nel 2015 per 5 miliardi di dollari. Questi signori non avrebbero fatto i miliardi se fossero rimasti a fare Amiga asm, d'altronde alcuni di loro non avrebbero fatto i miliardi se non avessero precedentemente conosciuto l'asm.

Sono certo che non leggete il corso per fare soldi con l'asm, ma per provare a sentirvi con poteri retro-sovrannaturali, e chi sa, indirettamente la rara conoscenza di asm che opera in background, potrebbe essere la differenza che vi fa fare i miliardi con altre tecnologie contemporanee. Infatti spesso mi scrivono sconosciuti che fanno di tutto tranne l'asm Amiga, ringraziandomi che tot anni fa, iniziarono proprio con questo mio corso che, se chiedete a un accademico, vi dirà insegna gli "anti-pattern" ovvero il contrario di come bisognerebbe programmare nel 2016.

Stefania *Yuki* Calcagno

Nel 1989 in un uggioso pomeriggio invernale eravamo a casa di Paolo *Levin*, a Savona. A quel tempo avevamo entrambi nick differenti, ma in questa prefazione userò solo quelli nuovi, *Yuki* per me e *Levin* per Paolo. Insieme decidemmo di formare un nuovo gruppo Amiga, completamente nuovo per la scena Italiana. Con le nostre BBS, con i propri coders, grafici, swappers, musicisti. Io ero una 'coder', Levin un 'sysop'. Entrambi già conosciuti e con un passato nel Commodore 64 e in altri gruppi Amiga.

L'idea era di fare un gruppo tutto Italiano che potesse competere con i più strutturati e famosi gruppi Nordici (Svezia, Norvegia, Danimarca, Finlandia e Germania erano il Top all'inizio della scena Amiga). In realtà la cosa ci sfuggì un poco di mano e diventammo uno dei più produttivi e famosi gruppi mondiali, con sezioni in USA, Francia, Germania, Svezia, Norvegia, Svizzera e membri addirittura in Turchia e Ungheria.

Quel pomeriggio io e Paolo *Levin* decidemmo di chiamare questo gruppo **Ram Jam**. Perché suonava bene *"Marmellata di Memoria ad Accesso Casuale"*, tradotto da un gioco di parole in Inglese. E anche perché la musica Indie e un po' new wave dei Ram Jam, gruppo musicale Inglese degli anni 70-80, che cantavano la splendida Black betty ci piaceva non poco!

Tra i primi membri, tutti liguri: eravamo amici e vicini all'inizio dell'avventura, ricordo Roberto *Bar*, Andrea *Executor* e Piero *Grizzly*, il mitico Sergio *Deus* unico non ligure della prima guardia, capostipite dell'avvento della mitica sezione Marchigiana – Abruzzese – Molisana, che ha contribuito a scrivere pezzi di storia dell'Amiga! Fabio *Randy*, l'autore di questo fantastico corso di Assembler, scritto con un editor di testi ASCII su un Amiga nei primi anni '90, si unì poco dopo alla ciurma.

Fabio è stato sicuramente uno dei migliori coders della scena Amiga Italiana e non, e sicuramente il più conosciuto tra i RamJammers. Insieme ad Andrea *Executor*, Pietro *Darkman*, Danilo *M.a.s.e.*, Fabio *Maverick*, Piergiorgio *Zibri* e Massimo *Mr.Buck* (i nostri coders Italiani, questo è un libro Italiano, non me ne vorranno i nostri fratelli in giro per il mondo!), Fabio *Randy* ha contribuito in modo importante alle produzioni Ram Jam ed a rendere la scena Amiga, Italiana e no, un posto bellissimo!

Qualche tempo fa, ad Alessandro *Trantor* venne in mente una magnifica idea: riprendere il corso di Assembler degli anni 90 di Fabio e trasformarlo dall'originale ASCII Amiga in un formato impaginabile e stampabile. Sia per preservare questo lavoro magnifico, che per dare la possibilità a "nuovi adepti" di studiare l'assembler Amiga.

Nel 2016. Si, nel 2016, proprio oggi tante persone vorrebbero cimentarsi a scrivere giochi o demo con questo vecchio codice macchina Motorola. Complesso, difficile, poco intuitivo ma, come dice Fabio, che da poteri sovrannaturali. Altri RamJammers hanno dato una mano a questo progetto, chi poco chi tanto: Nicola *HanTareX*, Giacomo *Jack Tory*, Dario *Rio*, Christian *DaGoN*. È grazie a iniziative come questa che la leggenda dell'Amiga continua a sopravvivere all'aumento di capacità di calcolo, di potenza grafica, di dimensione sia fisica (in piccolo) che logica (in grande) dei supporti di memoria.

Grazie a Fabio per averci raccontato negli anni 90 come sfruttare al massimo le capacità dei chip Amiga e per aver incarnato lo spirito hacker e di Ramjam *"knowledge is not a crime"*; e anche per aver dato il permesso di ripubblicare l'intera opera in versione moderna e renderla disponibile gratuitamente alla collettività.

Grazie ad Alessandro che ha fatto un immane lavoro per mettere insieme tutti i pezzi e renderli di nuovo fruibili da tutti.

Grazie a tutti quelli che hanno comunque collaborato, anche solo per esserci stati vicini!

Amiga will never die.

Stefania Yuki *Calcagno*

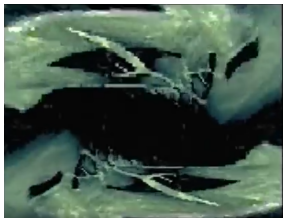
Massive Killing Capacity

Una delle più belle Demo della storia Italiana Amiga è stata sicuramente Massive Killing Capacity, per Amiga 1200 base, inespanso. L'ultima Demo di Fabio Randy per Amiga. Il suo testamento in Assembler. Lasciata ai posteri. Fondamentalmente è l'ultima e massima opera, nonché testamento spirituale di Fabio Ciucci.

In quest'ultima prefazione riportiamo alcuni screenshots di quella Demo, che puo' essere visualizzata tutta su You Tube cercando *Ram Jam Massive Killing Capacity* o semplicemente inserendo questo link: https://www.youtube.com/watch?v=vQqNDGJUm_8.

Per chi vuole è anche possibile scaricarla da qui: http://www.ramjam.it/ramjam/down.htm ed eseguirla direttamente sul proprio Amiga o su un emulatore come FS-UAE.

Probabilmente neanche alla fine del corso sarete in grado di fare effetti come questi, ma sicuramente avrete le basi per poter cercare di fare una Demo o anche un videogioco su Amiga. Che era ed è lo scopo del corso di Assembler di Fabio. Cercare di rendere possibile a tutti programmare questa splendida macchina che resiste ancora oggi a 20 anni dal fallimento di Commodore.



Resiste con una comunità enorme che continua a sviluppare tool, utility e giochi. Resiste con un numero incredibile di videogiocatori che adorano i titoli usciti per Amiga, la fluidità e la bellezza dei giochi stessi. Resiste, tant'è che oggi nel 2016, comprare un Amiga reale costa quasi di più di quando uscì sul mercato alla fine degli anni 80. Resiste con la scena che è sopravvissuta a se stessa e continua a fare i Demoscene party: ogni anno al **Revision**, in Germania c'è ancora una gara (*compo* in gergo, o *competition*) di Demo Amiga.

Anche Massive Killing Capacity fu presentata a un Demo Party, con relativa Competition. All'Assembly a fine 1996 In Finlandia. E successivamente in Italia al Gathering ad inizio 1997.

Buona lettura.

Stefania Yuki *Calcagno e Fabio* Randy *Ciucci*

Parte I

Teoria

CAPITOLO 1

LEZIONE 1

Per tutti coloro che hanno provato ad imparare a fare demo o giochi che sfruttino l'hardware di Amiga direttamente, ma non ci sono mai riusciti perché i libri erano scritti in maniera astratta e astrusa e i sorgenti di esempio, i listati cioè, erano poco commentati o troppo difficili, oppure per quelli che non ci hanno mai provato e si chiedono come si fa.

Devo ringraziare e salutare tutti coloro che hanno contribuito materialmente o moralmente alla realizzazione di questi due dischetti, in particolare:

- Luca Forlizzi (The Dark Coder)
- Andrea Fasce (Executor/RAM JAM)
- Sirio Zuelli (PROXIMA DESIGN)
- Alberto Longo (VIRTUAL DREAMS)

Nonché coloro che hanno testato le lezioni verificando se capivano o meno:

- Andrea Scarafoni
- Federico "GONZO" Stango, e altri.

Nella mia carriera di programmatore hobbista posso vantare la realizzazione di alcude demo/intro per delle BBS, ad esempio "`AMILINK.EXE`", per la banca dati AmigaLink, oppure per dei Club, come quella per il nuovo "*Amiga Expert Team*". Le mie "opere" maggiori sono la mia prima demo per il chipset AGA, il "*World of Manga*", che è stata pubblicata anche su alcune riviste, e il "*Naos*", che ho programmato per il gruppo *Nova ACIES*.

Devo precisare che sarebbe bene sapere almeno un poco di DOS prima di accingersi a leggere il mio corso, se non altro per sapere come salvare i listati! Dovreste aver trovato un manuale assieme all'Amiga... Comunque, in breve, nei dischi (sia Hard che Floppy) i dati sono immagazzinati in "file", ossia una serie di numerini uno dopo l'altro, che insieme possono formare file grafici, musicali, eseguibili, listati eccetera. Da notare che un dischetto vergine per poter essere

utilizzato deve essere *formattato*, altrimenti è impossibile scriverci. Una volta formattato, ci si può salvare qualsiasi file, sia figure con programmi di grafica, che testi (come questo che state leggendo) che altro. Un file si può copiare da un disco ad un altro, si può cancellare, oppure gli si può cambiar nome, eccetera. In un disco ci possono molti file, fino a che non si riempiono gli 880Kb circa, magari con 2 file da 400Kb o con una trentina di file più piccoli. Da notare che all'interno di un disco, per fare un pò di ordine, si possono generare varie "subdirectory", ossia dei "cassetti" più piccoli, delle divisioni in cui mettere i file. Ad esempio si possono generare le subdir DISEGNI e TESTI, in cui copieremo o salveremo rispettivamente delle immagini e delle lettere per la fidanzata, in modo da non mettere disegni e testi insieme sciolti nella dir principale. è come se il disco fosse un armadio, e le subdir dei cassetti di questo armadio. Dato che si possono fare delle subdir dentro le subdir, ognuno di questi cassetti può contenere file sciolti o "scatole" con file o altre "scatoline" più piccole dentro. Dunque un sistema simile a quello dei mobili! Per eseguire le operazioni tra i file si può usare la *CLI/Shell*, in cui occorre scrivere dei comandi come:

Dir Elenca i file e le subdir contenute in un disco

Copy Copia i file

Delete Cancella un file (ATTENZIONE AD USARE QUESTO!!!)

Makedir Crea un "cassetto" (o subdir)

Oppure si può agire col mouse da WorkBench, dove i file sono "raffigurati" come icone e le subdir come cassetti. Da notare che il drive interno si chiama "df0:", quelli esterni "df1:", "df2:" eccetera. L'hardisk di solito si chiama "Dh0:" (o "Hd0:"). Un sistema più veloce è quello di usare utility come DiskMaster o DirOpus.

Dunque quando avrete scritto qualche listato, lo dovrete salvare in un disco formattato, o sull'Hard Disk in quale subdir. Altra cosa da sapere è come si fa a fare un disco "autoboot", ossia che parte automaticamente inserendolo nel drive all'accensione o dopo un reset. Supponiamo di aver salvato un nostro programma *eseguibile* in un dischetto, col nome "mioprogramma". Occorrerà fare una subdirectory "S", in cui salvare un file di testo col nome di `startup-sequence`, in cui sia scritto il nome del programma da caricare automaticamente:

```
mioprogramma
```

La *startup-sequence* si può scrivere (editare) anche con il programma con cui state leggendo, che fa anche da text-editor. Ultima cosa, occorre "installare" il disco in questione, digitando da cli/shell il comando:

```
Install df0:
```

Oppure `install df1:` se si inserisce il dischetto nel drive esterno. Detto questo, si può continuare con le note.

NOTA: Se volete installare il corso sull'hard disk, ricordatevi di copiare nella vostra directory `S:` il file "`TRASH'M-ONE16.pref`" che si trova nella directory `S:` del disco del corso.

NOTA2: Se volete stampare i listati, considerate che sono compressi col powerpacker, per cui vi serve il *PowerPacker Patcher*, quello usato in questo corso. (il file è quello chiamato `PP` nella directory `C`). Per installarlo, basta avere in `LIBS:` la `powerpacker.library` ed eseguire il comando `PP`. I listati saranno autoscompattati al caricamento.

In questo corso verranno trattati i vari argomenti della programmazione, come il *COPPER*, gli *SPRITE*, il *BLITTER*, nonché il nuovo chipset AGA e la programmazione della scheda video

PICASSO II. Nel disco 1 gli argomenti sono: 68000, copper, playfields e sprites. Il blitter, l'AGA e il resto sono nei dischi 2 e 3, non del tutto terminati.

Per quanto riguarda la distribuzione e la copia di questo corso, dovete sapere che è GiftWare/-Shareware e non propriamente di pubblico dominio. Con questo intendo che potete copiare ai vostri amici questo corso senza problemi, basta che non lo VENDIATE per soldi, dato che i diritti su questo corso sono dell'autore, cioè me, e non certo del primo furbacchione che vuole speculare sul lavoro altrui. D'altronde, se è vero che lo potete copiare e distribuire AL SOLO PREZZO DEI DISCHI VERGINI, dovete anche ricordarvi che se seguite con successo le varie lezioni, riuscendo a programmarvi qualche cosa, avete tratto giovamento dal mio lavoro, per cui **dovete** ringraziarmi in qualche modo, specialmente se diventate i programmatori più ricchi del mondo (beh, nell'eventualità... Questo ringraziamento è quantificabile a vostro piacere, preferisco biglietti da 10.000. L'eventuale afflusso di regalucci o, meglio, vile denaro, mi incoraggerebbe a proseguire l'hobby della programmazione Amiga, e anche a fare nuovi capitoli del corso.

Mi farete anche un grande favore se copierete a tutti i vostri amici questo disco 1 del corso, anche se a voi personalmente non interessasse, perché darete la possibilità a qualcun altro di averlo e di imparare a programmare. Ho deciso di scrivere un corso di ASM (assembler) perché 10000 persone me lo hanno chiesto, e considerato che lo faccio per divertimento l'ho scritto in maniera molto discutibile, ma, a mio avviso, risulterà più chiaro ai principianti i quali, una volta iniziato a capire, potranno continuare più approfonditamente. Chi è già esperto di ASM troverà divertenti le lezioni, magari ci troverà delle inesattezze, perciò gli consiglio di consultare direttamente i listati di esempio: questo corso è per chi parte da zero. Infatti, dalla mia esperienza personale e da quello che mi dicono gli aspiranti "CODER" (in gergo programmatori CATTIVI), il problema è proprio capire il tutto e fare i primi due o tre programmini, dopodiché si diviene in grado di continuare da soli. Mi propongo, dunque, di insegnare a far girare delle palline per lo schermo o a farci saltellare una scritta a chi non sa nemmeno cosa sia il 68000. Se poi costoro vorranno diventare programmatori di giochi ed entrare nel TEAM 17 basterà che continuino.

> *PER IMPARARE A PROGRAMMARE UN GIOCO TIPO GODS O PROJECT X O COMUNQUE UN GIOCO CHE NON SIA UN SIMULATORE DI VOLO O UNO 3D, CHE INSOMMA NON COMPRENDA CUBETTI CHE RUOTANO, TUNNELL SINUSOIDALI O DISTORSIONI PROSPETTICHE, FRATTALI O TEXTURE MAPPING, GARANTISCO CHE BASTA AVERE LE COGNIZIONI DI MATEMATICA DI TERZA MEDIA.*

Con questo voglio togliere dalla testa a tutti che la programmazione assembler dell'Amiga sia piena di matematica. *Io credo invece che non c'entri nulla*. Se si intende fare un programma di matematica, si deve conoscere la matematica, come se si vuol fare un gioco del calcio bisogna conoscere il calcio.

L'importante è conoscere come funziona l'Amiga, il suo processore (nel caso dell'Amiga un Motorola 68000) ed i suoi chip custom (ossia quelli dedicati a fare la grafica ed il suono). Personalmente ho fatto le superiori all'Istituto d'Arte della mia città, ed ho imparato a fare cosucce in ASM già quando ero alle medie, quindi basta usare bene il tempo che si tiene acceso l'Amiga, anziché giocarci: non serve frequentare la facoltà di informatica all'università, dove non insegnano certo a programmare giochi o demo sull'Amiga!!!

Ma perché imparare a programmare giochi o demo? E cosa sono le demo? Dunque, i giochi cosa sono lo sanno tutti, quindi si suppone che chi voglia imparare a programmarli si sia stancato di vedere giochi che non sono come vorrebbe, e si vuole fare il "SUO", come vuole lui, pixel per pixel. Per quanto riguarda le demo invece occorre fare una breve spiegazione. Demo sta per "demonstration", ossia dimostrazione grafica. Dimostrazione di cosa? Della potenza dell'Amiga e della bravura dei programmatori, naturalmente.

Comunque c'è qualcosa di più: *LA SCENA*. Non quella del teatro, ma l'*"AMIGA SCENE"* (in inglese, la lingua ufficiale della scena stessa). Immaginatevi la scena della musica: ci sono vari gruppi con cantanti, batteristi, eccetera. Per l'Amiga, invece, troviamo vari gruppi con *coder* (programmatori), *GFX artist* (grafici), *musicians* (musicisti), che invece di fare un "VIDEO" come quelli che fanno i gruppi della scena musicale come loro contributo, fanno una *"DEMO"*, che si aggiunge alle altre fatte da altri gruppi in tempi e luoghi diversi. Ci sono poi gli *swapper* e i *trader* che sono rispettivamente coloro che scambiano e distribuiscono le demo via posta o via modem... costoro non producono niente, ma hanno una importanza nella scena, perché una cosa che non circola è come se non ci fosse. D'altronde, costoro aspirano a diventare *coder*, *grafici* o *musicisti*, per contribuire a fare una DEMO, anzichè scambiare opere altrui.

Ci sono molti gruppi nell'*Amiga Scene*, che hanno membri in tutti il mondo, in particolare in Europa. I nomi di alcuni gruppi più famosi sono *ANDROMEDA*, *BALANCE*, *COMPLEX*, *ESSENCE*, *FAIRLIGHT*, *FREEZERS*, *MELON DEZIGN*, *POLKA BROTHERS*, *PYGMY PROJECTS*, *RAM JAM*, *SANITY*, *SPACEBALLS*... Da notare che ogni membro della scena si fa chiamare con uno pseudonimo, detto "handle". Insomma, un nome d'arte: per esempio due coder degli ANDROMEDA si fanno chiamare "Dr.Jeckyll" e "Mr.Hyde", uno dei FREEZERS si fa chiamare "Sputnik", poi altri di vari gruppi sono: Hannibal, Dan, Paradroid, Dak, Wayne Mendoza, Performer, Bannasoft, Laxity, Vention, Psyonic, Slammer, Tron, Mr. Pet, Chaos, Lone Starr, Dr. Skull, Tsunami, Dweezil... Il nome completo si indica con l'handle seguito dal gruppo di appartenenza, ad esempio *CHAOS/SANITY*, *DWEEZIL/STELLAR*, *DAK/MAD ELKS*, e così via. Io, per la scena, sono *RANDY/RAM JAM*, ma ovviamente Fabio Ciucci per chi rimarrebbe perplesso, non conoscendo l'argomento. La scena organizza dei *PARTY*, delle specie di feste-ritrovo, dove i gruppi portano la loro demo, e ci sono delle competizioni con votazioni e premi anche di milioni per i vincitori.

Alcuni coder di demo poi passano a fare i giochi, dato che l'argomento è sempre quello. Ad esempio il programmatore di Banshee è *HANNIBAL/LEMON*., quello di Elfmania è *SAVIOUR/COMPLEX*, quelli di Stardust sono *DESTOP/CNCD* e *SCY/CNCD*, e la lista potrebbe continuare... Comunque nel disco 2 è presente una lezione solo sulla SCENA.

Ritornando alla programmazione assembler, sia che vogliate fare demo o giochi, vi sconsiglio di cominciare ad imparare studiando i listati di *routines* 3d (routine=parte di un listato o programma), perché sono le più complesse, che io stesso digerisco male, non per la programmazione in sé ma per le formulacce di matematica che contengono. Ma attenzione! Non dovete nemmeno pensare che se non serve la matematica servano conoscenze di elettronica o che sia necessario studiare gli schemi elettrici di Amiga!!! Quello va fatto solo se volete fare un programma per gestire una scheda grafica o un digitalizzatore video o simili.

Vi assicuro che potete, ad esempio, far apparire sullo schermo una figura o suonare una musica senza conoscere di dove passano i fili!!! Conosco persone che hanno imparato l'assembler a 12 anni e altre che lo hanno imparato a 30 o 40, senza conoscere bene la matematica e senza conoscere l'inglese. Quindi nemmeno l'età è una scusa accettabile per non provare! *Già! Perché dovete togliervi dalla testa anche che è indispensabile la conoscenza dell'inglese!*

Devo ammettere, però, che la conoscenza dell'inglese può rendere più facile il tutto, perché i comandi ASM sono abbreviazioni di parole inglesi, tipo *SUB* e *ADD* che significano *SOTTRAI* e *ADDIZIONA*. La conoscenza del *WorkBench* e dell'*AmigaDOS* non vi saranno utili per la programmazione in se, in quanto il computer in realtà funziona molto diversamente. Io direi, in maniera più semplice, che queste "sovrastrutture" sono il sistema operativo, localizzato nel chip del *KickStart*, senza il quale all'accensione non comparirebbe nemmeno la schermata che chiede di inserire il dischetto. Le finestre che vedete e spostate sono il frutto di migliaia di linee di codice ASM, contenute nel KickStart, infatti basta vedere la differenza delle finestre tra il Kick 1.3 e il Kick 2.0, che non sono dovute alla differenza dei dischi inseriti, ma alle differenze nel Kick stesso.

Se volete fare programmi stile *DeLuxe Paint*, gestione casalinga, word processor, o comunque utility per WorkBench che aprano le loro finestrelle su cui selezionare i gadget e i menu a tendina, vi consiglio di imparare il linguaggio C anzichè l'ASM, in quanto è più indicato e una volta imparato potete convertire i vostri listati facilmente all'ambiente MS-DOS e WINDOWS, nel caso che voleste abbandonare l'Amiga.

Se invece siete affascinati dalle demo grafiche con le palline rimbalzanti e le scritte metallizzate e sognate di programmare giochi tipo AGONY, LIONHEART, SHADOW OF THE BEAST, TURRICAN, APYDIA, PROJECT X, SUPERFROG, ZOOL, GODS, CHAOS ENGINE, XENON II, LOTUS ESPRIT, e mettiamoci anche SENSIBLE SOCCER, sia chiaro che si possono fare solo in ASSEMBLER PURO!!! E non richiedono particolari conoscenze di matematica: bastano le classiche addizioni, sottrazioni, moltiplicazioni e divisioni, e qualche tabella di SENI e COSENI per fare, ad esempio, le palline che cadono con una traiettoria a parabola, o comunque seguendo una curva: queste tabelle non sono altro che una serie di numeri in memoria tipo 1, 2, 3, 5, 8, 10, 13, 15, 18, 23 che sono, ad esempio, la progressione della posizione orizzontale e un'altra serie di numeri che sono la progressione della posizione verticale; queste serie di "tabelle" o *SINUSTAB*, cioè una serie di numeri che definiscono le coordinate di una curva, possono essere costruite con un apposito comando, il CS, presente nell'ASMONE, l'assemblatore, anche senza conoscere esattamente la trigonometria, può bastare sapere i parametri da passare e fare delle prove.

Di queste SINUSTAB o TABELLE ce ne sono molte nei giochi e nelle demo, in quanto molti movimenti ondeggianti non sono calcolati del tutto sul posto. Se invece sognate di fare delle *adventure* tipo *Monkey Island*, o dei giochi manageriali, in cui appaiono cioè solo schermate grafiche ferme con qualche ometto che ci si muove dentro lentamente, in cui il gioco consiste nel selezionare con il mouse degli oggetti o delle scritte, allora si può usare anche il linguaggio C, perché il gioco potrebbe essere convertito facilmente su PC, dove si farebbero un bel po di soldoni. D'altronde il C del PC lo insegnano nelle scuole scientifiche, e molto bene nelle università informatiche, e i soldoni li faranno loro.

NOTA: conoscere l'assembler dell'Amiga può rivelarsi utile se si passa, in seguito, a programmare anche un altro tipo di computer con lo stesso microprocessore, ossia il Motorola 68000 che, per fare un esempio, è usato da computer quali Apple MacIntosh e Atari ST.

Questi computer hanno però diversi sistemi operativi (diversi dal KickStart Amiga) e diversi chip dedicati alla grafica ed al suono, dunque vi servirà la conoscenza delle istruzioni del 68000, ma non quella del sistema operativo Amiga e dei suoi chip grafici, dovrete imparare da capo; d'altronde anche con linguaggi come il C dovrete imparare il nuovo sistema operativo.

Se ad esempio usate il linguaggio C e fate un programma per WorkBench che apre le finestre e magari fa dei disegni tipo montagnine, nel caso che compraste un PC MSDOS e voleste rifarvelo su WINDOWS, le parti del vostro programma inerenti ai calcoli per fare le montagnine e la struttura generale la potreste riutilizzare, ma tutta la parte inerente all'apertura delle finestre WorkBench e dei suoi gadget di selezione la dovreste buttare e sostituire con le istruzioni per Windows, e vi assicuro che imparare un altro sistema operativo e convertire un programma costa dei mesi o degli anni. NOTA: un programma scritto in assembler 68000 funziona benissimo sugli altri processori più potenti,a patto che si siano tenute presenti alcune cose.

Se state leggendo ancora significa che siete imperterriti. Allora completo la lista delle utilità dell'assembler... (il linguaggio in sé si dice *ASSEMBLY*, il programma che lo compila si dice *ASSEMBLER*, ma è uso comune chiamare assembler anche il linguaggio). Innanzitutto l'assembler rimane il linguaggio più veloce, specialmente se lo sapete bene, e la stessa cosa fatta con un altro linguaggio sarà sempre più lenta di una fatta in assembler.

Poi rimane anche l'unico mezzo per creare effetti grafici speciali, mai visti prima: potete ottenere effetti speciali anche con un titolatore, ma potete fare SOLO quelli definiti dal programma. Infatti non è difficile scoprire con che programma è stata fatta una titolazione o un effetto spe-

ciale; lo stesso vale per i DEMO MAKER, di cui il migliore è il *TRSI DEMOMAKER*, che ha degli effetti interessanti, ma ormai anche i bambini riconoscono una cosa fatta col demomaker, perché c'è sempre la scritta dorata sopra e sotto e al centro o le palline o le stelline... E BASTA!!! non se ne può più! Imparando a programmare in assembler, invece, si possono inventare degli effetti mai visti prima, perché non si è limitati a dover scegliere tra una ventina di effetti pronti per l'uso che altre migliaia di persone hanno usato, riempiendo reti televisive private e dischi. Per darvi un'idea dell'infinita varietà di cose che potete inventare in assembler posso nominare la SPACE-BALLS "state of the art" DEMO, una delle più conosciute, che non è difficile da programmare e ha stupito per le figure stilizzate di donne che ballano in mezzo a degli effetti speciali.

Se un programmatore ha più pazienza può anche programmare un gioco, dapprima per giocarselo, per il gusto di farsi il gioco dei sogni, per sperimentare i veri limiti dell'Amiga, per vedere quanti ometti si riesce a muovere senza rallentare lo schermo, poi nulla vieta di provare a fare un gioco commerciale, che richiede anche la collaborazione di grafici e musicisti, nonché tutta la parte relativa alla commercializzazione che spesso premia più la pubblicità fatta al gioco che la sua effettiva validità, a parte i casi in cui la validità è tanta che il successo arriva comunque.

Perché non mettersi a fare un gioco per CD32?? Basta fare un gioco AGA che sfrutti i 600MB di capienza del CD: per esempio un gioco in cui lo sfondo sia un "film" caricato in tempo reale, su cui far girottolare un RAMBO ammazzatutti o un'astronave. La difficoltà non sta nè nell'imparare il nuovo chipset AGA, nè nell'adattamento per CD, infatti il chipset AGA è molto simile a quello normale, basta impararsi qualche registro nuovo, e il processore funziona allo stesso modo, mentre per quanto riguarda la gestione del CD è ancora più facile, perché basta studiarsi i 2 dischi del "CD 32 DEVELOPER KIT" che circola tra i programmatori. Dunque l'assembler alle soglie del 2000 può essere ancora all'avanguardia, ovviamente per certi compiti in particolare, e se la tecnologia del 2000 sarà tutta su CD, come auspicano coloro che si comprano il PC per giocare ai giochi del CD o spesso per vedercisi le donne nude, dato che i CD sul PC MS-DOS sono in maggioranza slideshow sexy, anche l'Amiga avrà il suo software su CD, che potrebbe essere sviluppato da qualche tizio che un giorno cominciò la sua avventura leggendo un certo corso di programmazione...

Conoscendo come funziona il tutto, si può anche capire come funzionano certi programmi o giochi e si possono modificare delle sue parti: ad esempio si può capire come mai un gioco o un programma non funziona sui nuovi modelli Amiga e si può modificare per farlo funzionare, si possono fare certe modifiche ai programmi, per fare un esempio ho modificato una utility in modo che usasse la memoria virtuale su disco sull'Amiga 4000, altre volte ho velocizzato dei programmini PD, di cui ho "rubato" e velocizzato le parti più importanti. Infine si possono fare i cosiddetti trainer, le vite infinite, si trovano cioè le parti di listato che sottraggono una vita al povero PLAYER 1 e si modifica il tutto, magari facendo aumentare le vite quando si è ammazzati... per vedere e capire come funziona un gioco o un programma però è necessario conoscere VERAMENTE bene l'ASM e disporre di un monitor L.M o meglio di una cartuccia tipo ACTION REPLAY (Il monitor L.M. è un'utilità che permette di disassemblare, cioè visualizzare le istruzioni presenti in una sezione di memoria, e se si trova in che punto della memoria sono le istruzioni che tolgono una vita, si può modificare il tutto.

L.M sta per Linguaggio Macchina, cioè linguaggio del microprocessore, che è quello prodotto dall'assemblatore). Queste operazioni comunque sono una cosa difficile, e cominciare tentando di far diventare verde un ometto blu in un gioco non è certo utile. Ho visto tanti ragazzi sciupare il loro tempo andando a caso con i monitor L.M. e le cartucce tentando invano di fare non si sa cosa, cambiando le scritte nei programmi o nei listati, senza capirli, dicendo che li avevano fatti loro o che ci avevano fatto non si sa quali importanti modifiche. Costoro tutt'oggi non sanno visualizzare un'immagine in assembler; in gergo questi ciarlatani sono detti *LAMER*.

Facciamo il punto della situazione: se siete uno di quei diciottenni classici bianchicci e gobbi,

senza donne, e state andando a caso con i monitor LM per la memoria del vostro povero Amiga, vantandovi di essere un grande hacker, allora vi consiglio di posare il monitor e seguirmi per la retta via. Anche io ho cominciato in quella maniera ridicola (a 8 anni però! non a 18!), ma poi mi sono ravveduto e ho cominciato a leggere i libri senza saltare le pagine. Ecco un libro che vi potrebbe servire: *IL MANUALE DELL'HARDWARE DELL'AMIGA*, della IHT. Questo manuale spiega come funzionano i CHIP CUSTOM, quelli che fanno la grafica ed il suono dell'amiga, nonché come pilotare il DISK DRIVE eccetera.

Questo è indispensabile, ma per visualizzare anche una sola immagine bisogna conoscere anche il 68000, essendo il 68000 che gestisce i chip grafici. Inoltre il tutto rimane una cosa astratta, una specie di sintetica serie di tabelle di riferimento, e non ci sono validi esempi. Molti esempi li potete comunque trovare nel mio corso!!!! Se sapete l'inglese cercate l'ultimo *HARDWARE REFERENCE MANUAL*, che è aggiornato sui nuovi chip ECS. Comunque potete farne a meno per la durata del mio corso in quanto le cose principali ci sono, anche sui chip AGA dell'Amiga 1200. Inoltre nell'ASMONE è incluso un comando, *<=C>*, che da una spiegazione di tutti i registri *$dffXXX*, sia in generale che in particolare, ad esempio:

```
=C 100
```

vi darà una spiegazione del registro `BLTCON0`, concernente la risoluzione grafica, allo stesso modo *<=C 040>* vi darà un sunto del `BLTCON0`, reg. del BLITTER (*$dff040*).

I libri tipo *ROM KERNEL MANUAL* e *PROGRAMMARE L'AMIGA volume 1 e 2* della IHT, non sono utili alla programmazione diretta all'hardware, quella che tenterò di insegnare in questo corso, ma sono utili a chi voglia fare programmi per il workbench o l'amigados, che usino il sistema operativo contenuto nel kickstart e nei disks del workbench... programmi con finestre intuition dunque, non schermate con palline ed equalizzatori o ometti saltellanti tra le fiammelle... dunque più utili ai programmatori in C. *Non ve li consiglio...* programmare così è *noiosissimo*.

NOTA: Se, invece di essere utilizzatori bianchicci di monitor LM a caso, siete degli avidi ricercatori di giochi nuovi da copiare e da finire, passate ore al telefono a chiedere delle ultime novità, e le ore rimanenti a copiare con XCOPY e a giocare, magari sempre col trainer tanto per finire più in fretta, allora è peggio che essere gobbi col monitor LM: o interrompete questo affanno della ricerca e della copia, o rimarrete degli ipertesi che non sanno assolutamente come mai gli si muovono gli ometti per lo schermo, ne saprete mai come farvelo da voi un trainer al gioco col menu e tutto, e vi assicuro che quando vi siete fatto un trainer da voi, poi non vi interessa più di finire il gioco, ma piuttosto di capire come funziona.

Questa è la differenza che c'è tra il giocatore ed il creatore del gioco, tra il popolo assoggettato e stolto e i capi del regime che lo comandano facendogli passare notti insonni a finire (col trainer o meno) una miriade di giochi, non importa quali, basta che siano tanti e nuovi, copiati con l'XCOPY (che tra l'altro è il peggior copiatore al mondo! usate il DCOPY piuttosto!).

P.S: a proposito di donne, NON HO MAI VISTO NULLA PROGRAMMATO IN ASM DA UNA DONNA!!!! Se sta leggendo un rappresentante del sesso femminile, credo che questo sia un motivo in più per essere la prima!!! Una ragazza che, invece di interessarsi a pettegolezzi su persone sconosciute, o alle vetrine dei negozi, si metta a programmare robe pazzesche in gonna, credo che metterebbe in crisi di identità un bel po di bambinoni che si ritengono intelligenti facendo vedere, alle (poche) ragazze che conoscono, quanto muovono bene la freccia del mouse o le finestrine del workbench, pensando che tanto non capiscono nulla e che gli possono inventare che sono dei geni e che stanno avendo dei collegamenti con la NASA, quando invece non sanno nemmeno formattare un dischetto.

Vi anticipo che la `LEZIONE2.TXT`, che leggerete con i suoi sorgenti esempio dopo questa `LEZIONE1.TXT`, è la più **difficile**, quella **fatidica**, cioè se riuscite a passarla il gioco è fatto, perché

già dalla lezione 3 si fanno i primi effetti speciali col copper e procederete veloci come delle fucilate fino in fondo. Dunque vi chiedo di avere la pazienza di superare con calma e impegno la `LEZIONE2.TXT`, senza saltare niente.

Ora facciamo un'analisi dei programmi usati per programmare in assembly:

ASSEMBLATORE è il programma che traduce il listato fatto di comandi in formato simbolico (move, add...) nel suo equivalente binario (cioè in bytes). Cioè traduce un un testo, leggibile dal programmatore, nel formato reale delle istruzioni come le legge ed esegue il processore (una sequenza di numeri). Per esempio il comando `RTS` sarà trasformato in *$4e75*, e così via. Questo rende umano programmare, perché immaginatevi che roba programmare sapendo a memoria i numeri corrispondenti a ogni istruzione!!!! Programmare per NUMERI vorrebbe dire programmare in vero LINGUAGGIO MACCHINA, ossia L.M, ma è inutile, si fa molto meglio in ASSEMBLY, cioè usando delle parole convenzionali, dette COMANDI, al posto dei numeri reali. Questo codice binario risultante è chiamato codice oggetto ed è direttamente eseguibile dal computer, infatti può essere salvato il file eseguibile, oppure si può collaudare il programma. Ricordatevi che in assembler è in uso anche la numerazione esadecimale! I numeri esadecimali sono quelli preceduti dal *$*, e sono in base 16, come spiegheremo, e possono contenere anche le lettere *ABCDEF*, come in *$4e75*. Si deve tenere presente che se il listato ha degli errori "grammaticali" ci viene comunicato dall'assemblatore, infatti ci sono delle precise regole a cui attenersi: ad esempio le LABEL (o ETICHETTE), devono cominciare dall'inizio della riga, ossia non devono essere precedute da spazi, e devono terminare con i due punti (:). Ad esempio una LABEL corretta è

```
1 PIPPO:
```

Infatti il nome si da a piacere, e PIPPO va bene, perché non contiene simboli come = + - eccetera, non ha spazi che la precedono, e termina con i :. Le *label* sono nomi che si danno qua e là nel listato a delle cose, e servono per indicare quelle cose durante il programma, se per esempio si da nome `PIPPO:` a una certa serie di istruzioni, quando nel programma diremo che si deve eseguire PIPPO verranno eseguite le istruzioni sotto PIPPO, allo stesso modo possiamo mettere una label ad una figura o a una musica; la label dunque rappresenta l'indirizzo di memoria dove si trova, come il nome dei luoghi rappresenta la posizione di quei luoghi! Se voglio andare in Australia, vedrò una bella label `AUSTRALIA:` sopra di essa. Ricordatevi però che le label servono a noi per orientarci, ma quando l'assemblatore trasforma tutto, nel codice oggetto non ci sono label, solo i numeri corrispondenti alle istruzioni.

Ci sono poi le istruzioni, che invece devono essere SEMPRE precedute da degli spazi, meglio se da un TAB (che fa 8 spazi in un colpo solo, è il tasto sopra CTRL), e seguite dagli operandi, ad esempio:

```
1 PIPPO:
2     move.l  $10,$20
```

In questo caso `MOVE.L` è l'istruzione, mentre il primo operando è *$10* e il secondo è *$20*. Alcune istruzioni necessitano di un solo operando e altre di nessun operando, ad esempio:

```
1     clr.l  $10
```

Necessita di un solo operando. Istruzioni come `RTS` non necessitano di operandi. Infine ci può essere un commento, utile per ricordarci cosa si sta facendo con le istruzioni: il commento si può scrivere dopo un punto e virgola (;).

```
1  PIPPO:              ; LABEL, che rappresenta l'indirizzo di MOVE.L
2      move.l $10,$20  ; istruzione a 2 operandi
3      clr.l $10       ; istruzione ad 1 operando
4      rts             ; istruzione senza operandi
```

I commenti sono ignorati durante l'assemblaggio, quindi potete scrivere di tutto, basta che sia dopo i ;. Questa è la grammatica. Seguendo queste semplici regole il programma viene assemblato. Poi che faccia quello che deve fare o no dipende da voi!!!

EDITOR invece è un programma che serve per scrivere o modificare i testi, nel nostro caso per scrivere i listati, che non sono altro che dei testi, fatti di parole chiave (`MOVE`, `ADD...`) e commenti del programmatore (posti dopo i ;). I più potenti editor possono cercare, agganciare e sostituire caratteri. Solitamente ai listati assembly si da un nome che finisce con `.ASM` o `.S`, io personalmente preferisco `.S`, infatti quelli del corso finiscono in .S, mentre i testi da leggere finiscono in `.TXT`, ma il nome del file ovviamente non ha importanza per l'assemblatore, che lo carica comunque.

MONITOR non è in questo caso da intendersi come quello schermo su cui vedete le immagini dell'Amiga, ma un altro programma che permette di far vedere i contenuti della memoria, per esempio che numero c'è all'indirizzo *$100*, e così via. Solitamente i *monitor* hanno anche un *disassemblatore*, ossia il contrario dell'*assemblatore*, che ci permette di vedere la memoria come istruzioni, anziché come numeri, ossia traduce i numeri nei rispettivi comandi simbolici (`MOVE`, `ADD...`), in modo da rendere chiaro il funzionamento. Trasforma cioè il *linguaggio macchina* in *assembly*, ossia ricostruire le istruzioni assembly che ogni numero rappresenta, riportando il *codice oggetto* alla forma originaria che avete usato nel listato. Per riprendere l'esempio usato per l'assemblatore, trasforma *$4e75* in `RTS`.

DEBUGGER serve per collaudare il programma istruzione per istruzione, visualizzando gli effetti delle istruzioni ogni volta, e può indicare la causa del malfunzionamento del programma. Quindi consente di far eseguire il programma a pezzi, cioè definire fino a quando eseguirlo, per controllare la situazione, e poi riprendere l'esecuzione, per trovare ogni errore. Infatti *bug* significa *errore*, in gergo; in inglese significa *pulce pestifera*, infatti gli errori di solito sono difficili da trovare nel programma; con il debugger si può verificare in che punto si verifica l'irregolarità.

Alle volte il codice oggetto per poter funzionare effettivamente su un sistema operativo deve essere *linkato* con il *linker*, perché i file eseguibili non sono semplicemente il blocco di istruzioni che avete assemblato, ma hanno delle parti che permettono di farlo caricare in memoria dal sistema operativo. Questo vale per i file `.EXE` e `.COM` del PC MS-DOS e per i file eseguibili di qualsiasi altro sistema operativo, è per questo che un eseguibile per Amiga non viene caricato da un ATARI ST o da un MACINTOSH, che pure hanno un 68000, proprio perché il formato FILE è diverso. Gli Amiga in particolare hanno gli *HUNK*, e per trasformare il codice oggetto in un file con HUNK, che possa essere eseguito clickandoci col mouse o caricandolo dallo SHELL, bisogna linkarlo. Per fortuna molti assemblatori hanno il linker incorporato, per cui non occorre fare questo passaggio.

Ebbene, l'assemblatore incluso in questo corso, il **TRASH'M'ONE**, ha un EDITOR, un ASSEMBLATORE, un MONITOR/DEBUGGER e un LINKER!!! Ossia tutto in UNO!!!! è la versione modificata PD (Ossia liberamente copiabile?) dell'Asmone.

A proposito dell'editor, potete cercare un testo premendo contemporaneamente i tasti AMIGA *<destro+SHIFT+S>* oppure selezionando col mouse (*<tasto destro>*) l'opzione Search nel menù a tendina sotto la voce "Edit Funct."; a questo punto apparirà in alto a sinistra la scritta "Search for:", dove dovrete scrivere la parola (o le parole) da cercare. Può esservi utile intanto

per ritrovare il punto dove siete arrivati nella lettura: se per esempio volete smettere qua per oggi, potete segnarvi la linea dove siete arrivati, in questo caso la 549 (indicata in fondo a sinistra), oppure il potete anche ritrovare questo punto del testo cercando una sua parola, per esempio "Funct", oppure "caso la 549", oppure "cercando", oppure quello che vi pare.

Normalmente, avremmo dovuto scrivere il nostro listato con un EDITOR, e salvare il listato (detto SORGENTE) con un nome a piacere. Poi avremmo dovuto caricare l'assemblatore, da cui caricare il listato, assemblare (cioè trasformare dal testo a suo equivalente in L.M) e salvare il codice oggetto. Per collaudare il programma, ovvero controllare se funziona, avremmo dovuto eseguirlo dall'assemblatore, oppure linkarlo, rendendolo eseguibile, e farlo partire dal DOS. Per tornare a modificarlo avremmo dovuto cercare l'editor, ricaricare il listato, modificarlo, salvarlo, e rifare tutto l'assemblaggio. Sul PC MSDOS questo è quello che si deve fare, infatti ho rinunciato a programmarci, mentre sull'Amiga col multitasking si possono caricare insieme l'EDITOR, l'ASSEMBLATORE ecc. Come se non bastasse, qualcuno ha inventato il mitico SEKA, simile all'attuale ASMONE, che aveva editor, assemblatore e monitor insieme. Con l'evoluzione siamo arrivati al MASTERSEKA, poi all'ASMONE e infine alle molte versioni modificate dell'ASMONE dai più svariati programmatori hobbisti. I due più accaniti modificatori (bravi!!) sono i TFA, che hanno fatto il TFA ASMONE, e DEFTRONIC, che ha fatto questo TRASH'M'ONE. Ho scelto quello di Deftronic perchè è quello che ha meno BUG, infatti essendo modificati alla meglio questi ASMONE spesso assemblano fischi per fiaschi o si bloccano improvvisamente, ma non ci si può certo lamentare con loro che si divertono ad aggiungere opzioni senza guadagnare un soldo!

Il risultato finale è che vi potete scrivere il listato, poi premendo ESC passate all'assemblatore/monitor, da cui potete assemblare (con *<A>*), oppure vedere i contenuti della memoria, sia come numeri che come istruzioni DISASSEMBLATE, potete verificare il funzionamento del programma, e infine salvare direttamente il FILE eseguibile con *<WO>*. Per non fare confusione considerate che una cosa è salvare il listato, ossia il SORGENTE, che è un TESTO, un'altra è salvare l'eseguibile, che è un PROGRAMMA fatto di istruzioni nel formato FILE ESEGUIBILE. Il sorgente può essere scritto anche con un altro editor, come il CED, e poi potete caricarlo dall'ASMONE. Allo stesso modo un testo fatto con l'Asmone può essere caricato da un editor. L'Editor dell'Asmone quindi non è che un normale EDITOR inserito in un assemblatore, con cui potete scrivere anche una lettera per la mamma, oppure modificare la STARTUP-SEQUENCE di un disco (Chi non sa cos'è, per favore si legga il manuale dell'AmigaDos!).

Ordunque, procederò facendo chiarimenti e spiegando a modo mio come funziona il computer, per evitare fraintendimenti.

Quello che organizza tutto è il microprocessore 68000, la CPU, ovvero Central Processing Unit, insomma il Boss... Il processore esegue delle istruzioni, infatti ha un set di istruzioni ben precise che sa eseguire, e che esegue una dopo l'altra (di seguito), a meno che nel suo cammino non trovi l'istruzione di saltare ad eseguire più avanti o più indietro, o di fare un certo numero di loop (o cicli). Nomino per esempio alcune istruzioni: `MOVE`, che significa "copia un valore da un posto ad un altro", ad esempio "`move $10,$20`" muove quello che è in *$10* nella locazione *$20*, oppure `CLR`, che significa AZZERA: "`clr $10`" azzera la locazione *$10*... (per LOCAZIONE intendo un punto della memoria accessibile dal processore).

A proposito! Il processore opera sulla memoria! Facciamo una mappa. Quando le istruzioni operano con indirizzi minori di *$200000* si sta operando nella CHIP RAM, ossia: da *$000000* a *$80000* ci sono i primi 512k di CHIP, quelli dei vecchi a500 o a2000, mentre se la RAM continua fino a *$100000* significa che c'è 1 MB di chip RAM, come negli a500+, a600 o nei nuovi a2000, se la memoria CHIP invece è di 2MB, come negli a1200 o negli a500+ o a600 espansi, ad esempio, la chip va da *$000000* a *$200000*. Insomma quando il processore lavora su indirizzi minori di *$200000* ci troviamo in CHIP RAM, ad esempio:

```
1     clr.l $30000
```

```
2       move.l  $150000,$1a0000
```

Sono istruzioni che operano sulla CHIP RAM.

Quando invece operano su indirizzi da *$200000* in avanti, ci troviamo in FAST RAM, ad esempio un a500 vecchio con 1MB di memoria, divisa in 512k di CHIP e 512k di FAST ha la memoria divisa in 2 pezzi:

1. da *$000000* a *$80000* i primi 512k di CHIP RAM
2. da *$c00000* a *$c80000* 512k di FAST RAM.

Potete verificare con utility come SYSINFO i blocchi di memoria che avete.

Poi ci sono delle zone di memoria speciali, come quelle della ROM Kickstart, ossia di solito da *$fc0000* per kick 1.2 e 1.3 o *$f80000* per kick 2.0 o 3.0. La ROM, a differenza della RAM, non può essere sovrascritta, si può solo leggere, e non si cancella quando si spenge il computer.

Un importantissimo indirizzo è *$dff000*, in quanto quando le istruzioni operano su indirizzi che vanno da *$dff000* a *$dff1fe* vengono azionati i CHIP CUSTOM della grafica e del suono, infatti per azionare la grafica bisogna mettere i valori giusti in questi indirizzi *$dffxxx*, detti anche REGISTRI, proprio perché ognuno ha una funzione: provate a fare dalla linea di comandi (premendo *<ESC>* si scambia tra l'Editor e i comandi) il comando *<=C>*, e vedrete il riassunto di quei registri, con il numero, in cui 000 sta per *$dff000* e 100 sta per *$dff100*, e il nome, ad esempio *$dff006* è `VHPOSR`, mentre *$dff100* è `BPLCON0`. Questi indirizzi o si possono solo leggere, o si possono solo scrivere, per esempio *$dff006* si può solo leggere, e *$dff100* si può solo scrivere. Noterete una W o una R tra il numero e il nome: quelli che hanno una W sono quelli che si possono solo scrivere, (WRITE in inglese), quelli con la R si possono solo leggere (READ). Alcuni sono S (strobe) o ER (EarlyRead), ne parleremo in seguito quando li useremo.

Altri indirizzi speciali si trovano nella zona *$bfexxx*, ossia da *$bfe001* a *$bfef01*: si tratta di indirizzi collegati al chip CIAA, che si occupa di varie cose come fare da timer, ossia da cronometro, e di controllare le porte come la parallela (quella della stampante). Analoghi compiti li svolge il CIAB, connesso agli indirizzi *$bfdxxx*.

Quello che dovete ricordarvi in pratica è che quando vedete un indirizzo del tipo *$dffxxx* o *$bfdxxx* o *$bfexxx*, stiamo operando su un registro CUSTOM, causando cose come il cambiamento dei colori dello schermo, o la verifica dei movimenti del joystick o del mouse, o altro ancora.

Per quanto riguarda la memoria RAM, sia CHIP che FAST, non vi interesserà sapere a che indirizzo si trova ogni istruzione, perché l'assemblatore, come sapete, ci permette di usare le LABEL, al posto degli indirizzi: le metteremo solo nei punti utili, ci penserà l'ASMONE poi a mettere gli indirizzi reali al posto delle label. Potremo vedere dopo a che indirizzo sono finite le nostre istruzioni, se ci interesserà.

Continuiamo con gli esempi delle istruzioni. Ci sono comandi come `ADD` e `SUB`, che significano ADDIZIONA E SOTTRAI, ad esempio `SUB #10,ENERGIA` sottrarrà 10 al valore dell'energia; ci sono le moltiplicazioni e le divisioni con `MULS`, `MULU`, `DIVS` e `DIVU`, e le operazioni logiche `OR`, `AND`, `NOT` ed altre. `JMP` significa JUMP, ovvero salta ad eseguire ad una certa locazione (esempio `JMP $40000`), `JSR` invece significa esegui una routine ad una data locazione fino a che non trovi un `RTS`, ovvero "ritorna che è finita la routine", e l'esecuzione continuerà dopo il `JSR`; `BRA` fa la stessa cosa di `JSR` e `BSR` fa come `JSR`. `TST` significa TESTA rispetto a zero, ovvero controlla se una data locazione o registro è uguale a zero; questa istruzione o l'istruzione `CMP`, ovvero COMPARA qualcosa con qualcos'altro, è seguita di solito da un salto condizionato: `BEQ` e `BNE` ad esempio, che significano:

BEQ Salta a una certa locazione se è vera la condizione (*Branch if EQual*)

BNE Salta se non è vera (*Branch if Not Equal*). Si creano così delle diramazioni varie.

Facciamo un esempio stupido:

```
Principale:
    bsr CAMPANE          ; BSR fa saltare sotto la label CAMPANE, dopodiché
                         ; ritorna qua ad eseguire BSR aspettamouse
    bsr aspettamouse     ; Aspetta che sia premuto il MOUSE
    bsr PAVAROTTI
    rts                  ; torna all'asmone o al workbench

aspettamouse:
    controlla se il tasto del mouse è premuto
    se non è premuto vai a aspettamouse, ossia fai il girotondo fino a che
    non è premuto il mouse. (in questo caso si mette un "BNE aspettamouse")
    rts                  ; fine subroutine aspettamouse, torna sotto il BSR

CAMPANE:
    dindon               ; una routine che suona dindon
    rts

PAVAROTTI:
    AAAAAHHHHHHHHH       ; una routine che fa cantare Pavarotti
    rts

    end                  ; Indica la fine del listato, si può anche non metterlo.

(quello che viene scritto sotto l'END non viene letto nè assemblato)
```

ordunque, eseguendo questo ipotetico programma, si può dire che “Principale” è la routine, appunto, principale, che richiama 3 routines (parti del programma a cui viene dato un nome, ad esempio PAVAROTTI) in sequenza: all'inizio il processore salterebbe sotto `CAMPANE:` e suonerebbe le campane, poi trova un `RTS` e torna sotto `BSR CAMPANE`, dove trova un altro `BSR` che lo porta sotto `aspettamouse:` che è una routine che fa un ciclo fino a che non è premuto il tasto del mouse... il processore controlla miliardi di volte il mouse e se non è premuto ritorna sempre a controllare senza sosta; quando il mouse viene calpestato (premuto) la situazione cambia, perché si esce dal ciclo infinito `ASPETTAMOUSE`, e si arriva al suo `RTS`, ossia all'uscita, che lo fa tornare a `PRINCIPALE` sotto il `BSR aspettamouse` che abbiamo superato (il processore esegue sempre l'istruzione seguente, ossia sotto, e anche quando torna da un `BSR`, ossia dall'esecuzione di certe istruzioni messe in altro luogo) e trova l'ennesimo `BSR` che lo porta a far cantare pavarotti. Infine tornato dal concerto di Pavarotti trova un `RTS`, che lo fa uscire da `PRINCIPALE` e quindi torna all'asmone o al workbench: IL PROGRAMMA è finito.

Ora spiegherò meglio come si sposta il processore tra le varie istruzioni. Nel caso `BEQ label`, si può parlare di diramazione, infatti a questo punto si possono prendere 2 vie: immaginatevi proprio un albero, di quelli secchi senza foglie, una quercia secolare, con il tronco nodoso, che a un certo punto si divide in 2 rami, poi ognuno di questi 2 rami si divide in 2, e così via. Quando arriviamo al `BEQ` è come se fossimo una formichina che è partita dall'inizio del programma, ossia dalla base dell'albero, in cui c'è il nostro formicaio `START:`, e siamo arrivati alla prima *diramazione*: a questo punto o scegliamo di proseguire sul ramo destro o su quello sinistro. Questa scelta il 68000 la fa in base al risultato della condizione, sia essa un `CMP` o un `TST`:

```
INIZIO:              ; formicaio nell'erbetta
    tst.b LABEL30    ; Il byte della LABEL30 è= 0??? (condizione esempio)
    beq RAMODESTRO   ; se si, allora salta a RAMODESTRO
    ...              ; non è=0, allora eseguiamo il RAMOSINISTRO
                     ; (significa che il byte è un numero da $01 a $FF)
    (Istruzioni del RAMOSINISTRO)
    rts              ; Fine, usciamo: abbiamo percorso il RAMOSINISTRO e non
                     ; quello DESTRO

RAMODESTRO:
    ...              ; (Istruzioni del RAMODESTRO)
    ...
```

```
    rts                 ; Fine, abbiamo percorso il RAMODESTRO e non quello
                        ; SINISTRO
```

In questo caso una condizione di `TST` (confronta con 0) e di `CMP` (confronta il primo operando col secondo) seguita da un `BEQ` (se si, salta a. . .) o `BNE` (se no, salta a. . .), serve a scegliere se eseguire una certa serie di istruzioni o un'altra, se prendere una via o un'altra. Abbiamo già usato il `BNE` per fare un ciclo (o loop) in cui invece un certo numero di istruzioni sono eseguite ripetutamente fino a che non è verificata la condizione, per esempio il loop che aspetta che sia premuto il mouse. Il Ciclo può essere paragonato, anziché ad una formichina che sale un albero, ad un ROBOT che ha la pazienza di fare anche un miliardo di volte la stessa cosa senza stancarsi o scioperare, ad esempio:

```
VAI IN CUCINA, CONTROLLA SE LA TORTA è COTTA, SE NON è COTTA TORNA IN SALOTTO
E TOGLI LE PULCI AL CANE PER 30 SECONDI, DOPODICHE'
VAI IN CUCINA, CONTROLLA SE LA TORTA è COTTA, SE NON è COTTA TORNA IN SALOTTO
E TOGLI LE PULCI AL CANE PER 30 SECONDI, DOPODICHE'
VAI IN CUCINA, CONTROLLA SE LA TORTA è COTTA, SE NON è COTTA TORNA IN SALOTTO
E TOGLI LE PULCI AL CANE PER 30 SECONDI, DOPODICHE'
VAI IN CUCINA, CONTROLLA SE LA TORTA è COTTA, SE NON è COTTA TORNA IN SALOTTO
E TOGLI LE PULCI AL CANE PER 30 SECONDI, DOPODICHE'
VAI IN CUCINA, CONTROLLA SE LA TORTA è COTTA, SE NON è COTTA TORNA IN SALOTTO
E TOGLI LE PULCI AL CANE PER 30 SECONDI, DOPODICHE'...
```

Come si vede un essere umano si ribellerebbe a fare per il tempo necessario alla cottura di una torta un vai e vieni simile in continuazione, ma il 68000 non batte ciglio. Quando finalmente la torta è cotta, si verifica il `BEQ`, e il ROBOT salta alla routine `TOGLILADALFORNOSENZASCOTTARTIEMETTILAINTAVOLA:`.

Potete intuire che con delle diramazioni qua e là, anche dentro loop più o meno grandi, si può fare una struttura complicata che soddisfa qualsiasi tipo di necessità, basti pensare alla complessità dei programmi che simulano lo sviluppo di una città, che a seconda di migliaia di situazioni simula il comportamento dei cittadini. Tutto questo è possibile tramite diramazioni alle volte connesse o cicliche tra di loro.

I rami, ossia i pezzi di istruzioni eseguiti quando sono verificati i `BEQ` o i `BNE` che li richiamano, o semplicemente perché sono trovate durante il cammino del 68000, sono chiamate *routine* o *subroutine*, cioè pezzi di programma fatti di un certo numero di istruzioni che eseguono un dato compito, nel caso del ciclo del ROBOT, le istruzioni che fanno togliere la torta dal forno potrebbero essere isolate in una unica routine, che potrebbe essere eseguita ogni volta che è necessario togliere la torta dal forno. Infatti l'utilità delle routine e specialmente delle subroutine isolate sta proprio nel non dover riscrivere ogni volta che si deve togliere la torta dal forno la stessa serie di istruzioni, ad esempio.

Queste istruzioni le possiamo isolare, e metterle da parte dandogli un nome, assegnandogli cioè una *label* all'inizio, e decidendone una fine con l'istruzione `RTS`. Diamo una definizione alla parola *subroutine*:

dicesi subroutine di un blocco di istruzioni alle quali è stato dato un nome, facendola iniziare con una label*, ossia un nome a piacere seguito dai :, e finire con una speciale istruzione di ritorno, l'*`RTS` *(*ReTurn from Subroutine*), e solitamente si fa eseguire con l'istruzione* `BSR`*, seguita dal nome della subroutine; dopo aver eseguito il* `BSR`*, il processore tornerà ad eseguire le istruzioni sotto il* `BSR` *che hanno fatto eseguire la subroutine stessa.*

Questo si può paragonare al comandante di un sommergibile, che in questo caso è il programma principale, il quale dando gli ordini esegue delle subroutine, ad esempio immaginatevi

che il comandante abbia visto al periscopio una nave nemica, a questo punto farà un `BSR ArmaTeSiluri`, ossia darà il comando di armare i siluri. Fino a che la subroutine che arma i siluri non sarà eseguita non potrà procedere. Una volta avvisato che sono stati armati, il comandante, ossia il programma principale, continuerà la procedura: ossia dare `BSR DESTRA` e `BSR SINISTRA` al reparto macchinisti fino a che la nave non si trova sulla traiettoria dei siluri; questo lo potremmo paragonare ad un ciclo in cui c'è un `CMP NAVE, SILURI` seguito da un `BNE SPOSTASOTTOMARINO`, ossia: "la LABEL che contiene la posizione della nave è uguale al contenuto della LABEL che contiene la posizione che raggiungeranno i siluri?", se non ancora (`BNE`), allora spostati ancora, cioè torna alla routine che controllerà se siamo più a destra o più a sinistra, e di conseguenza esegui le subroutine `SINISTRA` e `DESTRA`.

Questo ciclo è simile a quello del ROBOT che aspettava che la torta fosse cotta, ma in questo caso invece di aspettare la cottura siamo noi che attivamente dobbiamo raggiungere la posizione esatta, come nel ciclo che aspetta il MOUSE siamo noi che lo dobbiamo premere per fermarlo. Eravamo rimasti al ciclo di allineamento: all'improvviso il comandante da il comando di lanciare i siluri! (`BSR FUORIUNO`, `BSR FUORIDUE`). *BOOOOOOOOOOM...* Ha funzionato...morti da tutte le parti, calzini galleggianti, vedove e orfani sparse per tutta la Germania (nei film muoiono sempre i tedeschi), un relitto in fondo al mare. TRANQUILLI! Era solo una simulazione al computer ben riuscita.

Se siete entrati nella logica del processore, il gioco è fatto. Tutto quello che vedete girare sul computer, sia un programma per le previsioni del tempo, una demo con cubi e palline, un gioco di azione, è fatto di pezzi di programma che sono eseguiti ciclicamente o sequenzialmente, a seconda dei responsi delle varie condizioni `TST`, `BTST`, `CMP`. Dunque ogni tipo di operazione e di decisione, di qualunque ordine di complessità, e fatto di un certo numero di condizioni semplici, considerando che ogni subroutine può essere fatta con altre subroutine più piccole, ad esempio `TOGLILATORTADALFORNO:`

```
1  TOGLILATORTADALFORNO:
2      bsr SpengiIlForno
3      bsr ApriIlForno
4      bsr PrendiLaTorta (tanto è un robot e non si scotta)
5      bsr PosaLaTortaSulTavolo
6      bsr RichiudiIlForno
7      rts
```

A sua volta le subsubroutine possono essere fatte di altre subroutine:

```
1  SpengiIlForno:
2      bsr VaiSull'interruttore
3      bsr GiraloVersoSinistra
4      rts
```

La maggior utilità delle subroutine sta nel rendere più chiaro il programma, dividendolo in parti logiche, e nella possibilità di farsi una raccolta di routines che possono essere usate per altri programmi, ad esempio se avete una routine che legge la posizione del joystick la potete riutilizzare in tutti i giochi che farete, con lievi modifiche se necessario, allo stesso modo la routine che suona la musica, o quella che fa camminare un ometto sul video.

Questo è per dare un'idea del continuo eseguire e girovagare a seconda delle condizioni vere o false del povero microprocessore. Quando saltellando qua e là c'è un errore, ad esempio salta in una zona con dati caricati male da disk o dove il programmatore ha fatto cilecca, allora appare il mitico *GURU MEDITATION*, o *SOFTWARE FAILURE* nella sua inquietante finestra rossa lampeggiante.

La memoria riscrivibile (RAM) può essere modificata, e si divide in CHIP ram e FAST ram, come già detto. La differenza è che la GRAFICA e i SUONI devono essere in CHIP RAM, mentre le istruzioni del processore possono essere sia in CHIP che in FAST. Per esempio l'Amiga 500 vecchio 1.2 o 1.3 ha 512Kb di RAM, ovvero mezzo mega, e se si espande si arriva ad un mega di

RAM, ma gli altri 512k sono FAST, è per quello che ad esempio con il DeLuxe Paint si finisce la memoria prima con 1MB diviso in 512k CHIP e 512k FAST rispetto ad un a500+ che ha invece 1MB tutto di CHIP: la memoria nel vecchio 500 avanza, ma è di tipo FAST e non serve ad aprire un nuovo schermo, quindi dice che non c'è memoria. Quando si programma se si prova a visualizzare grafica messa in FAST succede il finimondo, di tutto tranne visualizare quell'immagine. La memoria è fatta a blocchi di varie misure, ad esempio su un a500 vecchio i primi 512k di chip ram vanno dall'indirizzo *$00000* a *$80000* e i 512k di espansione da *$c00000* a *$c80000*: il sistema operativo sa dove sta la memoria e carica i programmi automaticamente nelle zone vuote, ad esempio caricando un programma dal WorkBench o dal CLI o SHELL i dati dal dischetto saranno trasferiti (grazie al kickstart) in memoria, a seconda che sia richiesta memoria CHIP o FAST, dopodiché il processore salterà al punto in memoria dove ha caricato, (o meglio copiato dal dischetto) il programma. All'utente rimane oscuro in che punto della memoria sia stato messo il programma e dove il microprocessore stia lavorando. Ho detto che la memoria chip nel vecchio 500 va da *$00000* a *$80000*, la memoria infatti è divisa in parti, come una strada con tante casine le quali abbiano il loro indirizzo: non a caso si chiamano indirizzi o locazioni di memoria (*address* in inglese): all'inizio della strada c'è la casa 0, che contiene un byte, la casa dopo ha l'indirizzo 1, che contiene un altro byte, e così via. è usato però il sistema di numerazione *esadecimale*, cioè in base 16. Questo non è un problema, perché da ASMONE si può convertire il numero in qualsiasi momento usando il comando `<?>`: facendo "`?$80000`" si avrà un 524288 in decimale, che corrisponde a 1024*512, cioè mezzo Kb o "KAPPA RAM", appunto 1024 bytes, moltiplicato per 512. *$100000* invece è il doppio, ossia un mega... provate `?$80000*2` ("*" ovvero "MOLTIPLICATO").

I numeri esadecimali sono preceduti dal dollaro, come hai visto, i numero decimali non sono preceduti da nulla, quelli binari da un %. Queste cose sono basilari: come per le distanze esiste il metro, il decametro ed il chilometro, per la memoria esiste il **bit**, il **byte**, la **word** e la **longword**. Il *bit* è la parte più piccola di memoria; il *byte*, composto da 8 bit, è una unità che ha il suo indirizzo. Il processore cioè può dire: muovi (o meglio copia) il byte che è nella casina in "via della memoria n10" nella casina in "via della memoria n16", in questo caso ha copiato gli otto bit che erano nel byte 10 (ovvero *$A* in esadecimale o HEXadecimal) nel byte 16.

Per evitare confusioni, facciamo l'esempio inequivocabile: i bit possono essere a 0 o ad 1; nel byte 10 i bit erano: *00110110*, nel byte 16 invece *11110010*, dopo il `MOVE.B 10, 16` il byte 10 rimane *00110110*, il byte 16 diventa *00110110*. Il `.B` al `MOVE` significa che viene mosso un *byte*, cioè la parte più piccola che si possa copiare. Si può anche fare un `MOVE.W` ed un `MOVE.L`, ossia muovere una *word* (.w) o una *longword* (.l), che non sono altro che: 1 word = 2 bytes, una longword = 4 bytes, ovvero 2 word. Allora se si fa un `MOVE.W 10, 16`, nel byte 16 verrà copiato il byte 10, nel byte 17 il byte 11, ossia viene spostato un blocco di 2 bytes. Nel caso di un `MOVE.L` vengono spostati 4 bytes, ossia: nel byte 16 il byte 10, nel 17 l'11, nel 18 il 12, nel 19 il 13. Facciamo uno schemino:

```
PRIMA DEL MOVE.L 10,16 ;  08/09/10/11/12/13/14/15/16/17/18/19/20
                                C  A  N  E        G  A  T  T  O

DOPO IL MOVE.L 10,16 ;    08/09/10/11/12/13/14/15/16/17/18/19/20
                                   C  A  N  E     C  A  N  E  O

Se facciamo MOVE.B 20,14 ;08/09/10/11/12/13/14/15/16/17/18/19/20
                                C  A  N  E  O     C  A  N  E  O
```

Nella nostra supposizione le locazioni 08, 09, 14, 15 erano azzerate, mentre le 10-13 e le 16-20 avevano un valore, qua delle lettere per esempio. Concludiamo con un `MOVE.W 8,10`:

```
;08/09/10/11/12/13/14/15/16/17/18/19/20
            N E O    C A N E O
```

Con 3 istruzioni abbiamo trasformato CANE GATTO in NEO CANEO!!!!! A parte gli scherzi, non proseguite a leggere fino a che non vi è rimasto impresso nella memoria cerebrale il funzionamento della memoria sintetica!!!! Fate un po di giochetti coi `MOVE.X`, che vi fa bene! Provate ad esempio a trasformare ANTANI in TANTI NANI con vari `MOVE`, oppure SBLINDO in DOBLONI, oppure RENULOZ in ZUZZURELLONE, eccetera. Ricordatevi che le **istruzioni** del processore devono essere ad indirizzi pari, tipo 2, 4, 6... ossia allineati a *word*, oppure va tutto in GURU.

Per togliere dubbi, nella memoria ci sono una serie di valori uno dietro l'altro, che possono essere istruzioni del 68000, o dati come ad esempio le *sinustab* prima citate, figure, suoni, scritte da visualizzare... le istruzioni in memoria non sono nella forma MOVE.B 10,16, quella è una versione DISASSEMBLATA, in memoria ad esempio quella istruzione occupa 10 bytes, ed è: *$13,$F9,$00,$00,$00,$0A,$00,$00,$00,$10*, in cui *$13f9* significa in grandi linee `MOVE.B`, `$0000a` è 10 in esadecimale e $10 è 16 in esadecimale... allo stesso modo ogni istruzione ha i suoi bytes, ad esempio l'istruzione `NOP`, ossia no operation, che non fa nulla, in memoria è *$4e71*. Anticipo che oltre che operare sulla memoria il processore ha a disposizione dei registri, denominati registri dati e registri indirizzi, che sono 16 e lunghi una longword ciascuno, chiamati `a0,a1,a2,a3,a4,a5,a6,a7` gli Address reg, `d0,d1,d2,d3,d4,d5,d6,d7` i data reg; sono dentro il processore e quindi sono molto più veloci le operazioni tra 2 registri di quelle tra 2 indirizzi di memoria, ad esempio `MOVE.L d0,d2` sarà più veloce di `MOVE.L $100,$200`; si preferisce quindi fare operazioni mettendo i numeri nei registri piuttosto che nella memoria, se possibile.

La ROM come già detto non si può scrivere, cioè un `MOVE` che scrive nella ROM non ha effetto: un `MOVE` su *$FC0000* o su *$f80000* non serve a niente. Si possono solo eseguire le *routine* contenute nel ROM. Ma **essendo il Kick diverso in ogni versione, mai si deve saltare al Kick direttamente**. Il sistema operativo è fatto in modo che le *routines*, ossia i singoli programmi presenti nel kickstart, possano essere chiamate nello stesso modo qualunque sia il kick e ovunque sia collocato in memoria: questo viene fatto tramite dei `JSR`, ossia dei *jump to subroutine* (Salta ad un indirizzo, dopodiché ritorna e continua da sotto il `JSR`), che sono fissi partendo però dall'indirizzo presente nell' indirizzo 4, in cui è sempre presente l'indirizzo da cui regolarsi per fare i giusti `JSR` per eseguire le routines del kickstart.

I programmi per aprire le finestre del workbench o per stampare caratteri, per leggere o scrivere un file su disco devono chiamare la routine presente nel CHIP del kickstart ROM ogni volta, passandogli ad esempio il nome del file da caricare o le dimensioni della finestra da aprire; invece quando un gioco o una demo "salta" il sistema operativo non vengono fatte chiamate al kickstart: ad esempio il noto *XCOPY* apre un suo schermo, ed appare evidente che toglie di mezzo il multitasking e non ha le finestrine ed i menù da tasto destro come i programmi da sistema operativo. Allo stesso modo un gioco come quelli che ho menzionato prima, come *SENSIBLE SOCCER*, funzionerebbe anche se dopo il boot(la partenza) si rimuovesse il chip del kickstart, in quanto non vengono chiamate routines per aprire finestre o caricare file: le cose che appaiono a video sono controllate una per una e i dati dal disco sono caricati non come files DOS, ma come tracce lette direttamente spostando le testine del DRIVE dando corrente o meno ai pin del cavo.

Appare chiara questa differenza? Tra i programmi o giochi che *usano* il sistema operativo, ossia richiamano continuamente routines nella ROM e mantengono il multitasking e le finestre, e gli altri prog. che non aprono finestre o le aprono in maniera diversa dal WorkBench, e non possono essere eseguiti insieme al Deluxe Paint, scambiando la finestra o spostandola in basso??

Insomma la ROM si preoccupa di dialogare con l'hardware per noi se glielo chiediamo, e fa un certo numero di cose prestabilite, mentre se decidiamo di dialogare NOI con l'hardware, possiamo fare tutto il possibile, sempre che ne siamo capaci!!!

Ordunque ci occuperemo di fare codice senza usare la ROM. Ma allora useremo solo il microprocessore? e come si fa a visualizzare un immagine o suonare una musica? con dei MOVE???? Ora entrano in gioco i **chip custom**!!! Questi *chip* si chiamano *Paula*, *Agnus* e *Denise*, inoltre ci sono altri 2 *chip* detti *CIAA* e *CIAB*. Questi furboni sono quelli che fanno suonare l'amiga e che gli fanno visualizzare tutti quei colori sullo schermo. La maggior parte dei registri in questione si trovano alla locazione *$dff000* fino a *$dff1fe*, altri riguardanti le porte seriali, parallele, e dei disk drives si trovano in zona *$bfexxx* o *$bfdxxx*.

Una volta imparate tutte le istruzioni del 68000 si possono costruire programmi grossi come case, ma se si sposta memoria qua e la non si visualizza né si suona nulla! col processore bisogna pilotare questi chip; uno principale è il **Blitter**, che si occupa di disegnare le linee, copiare pezzi di memoria come scroll o ometti in giro per lo schermo, riempire aree (i solidi 3d sono disegnati e riempiti con il blitter; il processore si occupa di calcolare le coordinate delle linee che poi il Blitter disegna).

Quello che però visualizza il tutto e che determina i colori è il **Copper**: per fare un esempio il *$dff180* corrisponde al colore 0 ed il *$dff182* al colore 1, mentre nel *$dff006* c'è la linea dove il pennello elettronico è arrivato nel disegnare lo schermo, che viene disegnato 50 volte al secondo: questi registri infatti sono a SOLA lettura o a SOLA scrittura, ad esempio nel *$dff180* si può mettere un valore, ma non si può leggere che valore c'è, lo stesso vale per il *$dff006*, sul quale non si può scrivere; per cambiare la posizione al pennello elettronico esiste comunque un apposito registro, così per molti altri. Nei registri *$bfexxx* si può controllare il disk drive o le varie porte, tra cui quella del mouse: ad esempio al bit 6 dell'indirizzo *$bfe001* corrisponde lo stato del bottone sinistro del mouse, se questo è premuto o no, e si può controllare col processore ed aspettare che sia premuto prima di uscire. Ed è questo il primo esempio di programmazione che puoi analizzare caricando `LEZIONE1a.s`, il primo sorgente del corso, che comprende insieme un ciclo con il 68000, l'utilizzo di un registro *$dffxxx* e di uno *$bfexxx*. (caricatelo in un altro buffer di testo come spiegato sotto).

Un breve accenno su come usare l'assemblatore, in questo caso ASMONE. All'inizio si deve selezionare se allocare memoria CHIP o FAST, è bene selezionare quella chip per i sorgenti del corso, a seconda di quanta ne avete, selezionate il numero di Kb, almeno 250. Per selezionare una directory o un drive usate il comando *<v>*, per esempio per andare nella directory delle lezioni fate un "V df0:LEZIONI", per andare nella directory dei sorgenti fate un bel "V df0:SORGENTI", poi per leggere il sorgente o la lezione usate *<R>*, e selezionatelo con la finestrella. Si può scambiare con *<ESC>* tra la funzione di editor e la linea di comandi; cioè premete ESC e potete scorrere o modificare il testo, ripremete *<ESC>* e tornate alla linea di comandi dove potete ad esempio ASSEMBLARE il listato con *<A>*, dopodiché per farlo eseguire dovete premere *<J>* (JSR!!).

Potete anche caricare simultaneamente 10 testi, siano essi sorgenti o lezioni, perché, quando siete in modo EDIT, quando cioè potete scorrere il testo con il cursore e potete cambiarlo, se premete F2 scambierete listato, e andrete al secondo, che in questo caso sarà vuoto: se premete nuovamente F1 ritornate al testo caricato prima: in questo modo potete, ad esempio, tenere nel buffer 1 (ossia listato 1 richiamabile con F1) la `Lezione1.TXT`, mentre nel buffer 2, selezionabile con F2, potete caricare il listato inerente alla lezione1, ossia `Lezione1a.s`. In seguito potete mettere lezione1 nel buffer 1, lezione2 nel buffer2, nel buffer 3,4,5 dei listati della lezione2, eccetera, quindi potrete consultare la lezione, poi premendo un *<F4>* o un *<F5>* verificare subito l'esecuzione di un listato, o ritornare a vedere una cosa della lezione1 che non ricordate, eccetera.

NOTA: per scorrere di pagina in pagina usate i cursori (frecce) più lo *<SHIFT>*, ossia, per chi non aveva il C64, il tasto grande sopra *<ALT>* con la freccia. Vi spiego che succede quando fate *<A>*: il listato (o sorgente) è in formato testo normalissimo, ed è fatto di parole chiave che sono i comandi o altri simboli che conosce l'assemblatore. . . per segnare un gruppo di istruzioni o una

"variabile", o l'inizio di una tabella, o comunque avere un riferimento di un preciso punto del listato, si fanno delle ETICHETTE o LABEL, che non devono avere spazi dall'inizio del bordo, e devono finire con i : (DUE PUNTI). Il nome della label è a scelta, ma non si deve dare un nome che sia uguale ad un comando 68000!!! esempio:

```
1 WAITMOUSE:                ; la label
2     btst #6,$bfe001       ; tasto sinistro premuto?
3     bne.s WAITMOUSE       ; se no torna a WAITMOUSE (ripeti il btst)
4     rts                   ; Esci
```

Vi ricordo che i comandi devono avere una spaziatura, in questo caso ho usato il *<TAB>* (Il tasto sopra *<CTRL>* e *<CAPS LOCK>*), che fa 8 spazi con un colpo solo... Notate che non vanno messi i : (due punti) finali al nome della label (o etichetta) quando viene richiamata, ma solo a se stessa. Dunque, una volta editato il sorgente, va ASSEMBLATO con *<A>*; questa operazione fa leggere all'ASMONE il testo, e lo trasforma in codice, cioè nei bytes che saranno letti dal 68000 ed eseguiti come istruzioni. Una volta assemblato, il suddetto codice è in un punto della memoria che si può vedere con *<=R>*, e con il comando *<J>* il processore salta a quel punto della memoria ed esegue il nostro programma. Se l'ASMONE trova un errore nel listato non assembla tutto fino a che non viene corretto l'errore. I sorgenti del corso funzionano anche con altri assemblatori come DEVPAC 3 e MASTERSEKA, con tutti i kickstart e con tutti gli Amiga, compresi quelli AGA come il 1200 o il 4000.

Se avete verificato il funzionamento di `Lezione1a.s`, caricate in un altro buffer di testo (quello *<F3>*, ad esempio) il file `LEZIONE2.TXT` con *<R>*.

Se mancasse memoria quando scambiate buffer, significa che avete selezionato troppa memoria all'inizio (al messaggio ALLOCATE), e non ne è rimasta per la RAM DISK. La prossima volta selezionatene meno.

CAPITOLO 2

LEZIONE 2

Avete capito esattamente come funziona il sorgente `LEZIONE1a.s`??? Se non lo avete capito allora siete da neuro, e dovete smettere il corso.

Ora andiamo ad approfondire il linguaggio del 68000. Ho voluto anticipare col primo sorgente il fatto che il processore serve grossomodo per organizzare tutte le cose, ma che da solo non fa che cambiare valori nella sua memoria; mettendo certi valori in aree particolari di memoria come *$dffxxx* o *$bfexxx* si da corrente ai piedini dei chip della grafica, del suono e delle porte, e di conseguenza si può, come nell'esempio precedente, cambiare il colore dello schermo, o leggendo queste locazioni sapere a che linea il pennello elettronico sia arrivato o se il bottone del mouse è premuto. Per fare un gioco o una demo è necessario usare un gran numero di questi indirizzi, detti REGISTRI, e quindi è necessario conoscerli almeno quanto il linguaggio del 68000, (`MOVE`, `JSR`, `ADD`, `SUB` etc.) con cui si impostano. Per la programmazione di questo tipo come ho gia detto non si usano le *librerie* del ROM kickstart 1.2/1.3/2.0/3.0 (Ovvero le sue routines, o sottoprogrammi che consentono di aprire una finestra workbench o leggere un file, ad esempio) cioè si usano pochissimo: ad esempio per evitare di far andare in guru il workbench o per disabilitare il multitasking. Ritengo necessario quindi in questa lezione n.2 di approfondire l'uso del 68000, una volta che si è capito quale sia il suo ruolo.

La cosa più importante da imparare sono i modi di indirizzamento del processore, più che i comandi in se, infatti una volta imparato quello, ogni comando usa la stessa sintassi per l'indirizzamento e basta sapere che cosa fa il comando. Abbiamo già detto che il processore opera sulla memoria che è divisa in locazioni o indirizzi, la cui unità di misura è il byte, e solitamente l'indirizzo è in formato esadecimale, cioè in un formato numerico diverso da quello decimale, avendo infatti base 16. Questo non è assolutamente un problema: mentre con i numeri decimali una sequenza di 30 numeri ad esempio fa: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 etc..., in esadecimale fa 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e etc..., cioè nei numeri esadecimali si trovano anche le prime 6 lettere dell'alfabeto come se a fosse un 10, b un 11 eccetera; per convertire un numero da esadecimale a decimale o viceversa basta usare il comando *<?>* dell'asmone: per esempio facendo *<?10000>* si otterrà *$2710*, il corrispondente valore in esadecimale (i numeri esadecimali cominciano col *$*, quelli decimali non sono preceduti da niente e quelli binari da un %). I numeri esadecimali

sono usati perché sono più vicini al modo di pensare del computer che è ovviamente *binario*, cioè composto di soli 0 ed 1. Come esempio per iniziare a capire i vari modi di indirizzamento del 68000 useremo il comando CLR, che azzera la locazione di memoria indicata:

```
1       CLR.B   $40000         ; vi ricordate la differenza tra .B, .W e .L?
```

Questa istruzione "pulirà", cioè azzererà il byte n. $40000, ossia l'indirizzo *$40000*. Questo è il caso più semplice, detto *assoluto*; cioè si indica direttamente in che indirizzo fare il CLR; nell'assemblatore sono in uso le LABEL, che servono ad identificare un punto del programma, in cui può esserci per esempio un byte da indicare: in questo caso invece di scrivere l'indirizzo si scriverà il nome della LABEL; l'assemblatore allora scriverà l'indirizzo effettivo del byte in questione: ad esempio, se modificassimo così il nostro primo listato:

```
1  Waitmouse:
2       move.w  $dff006,$dff180 ; metti il valore di $dff106 in $dff180
3                               ; cioè il VHPOSR nel COLOR0
4       btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
5       bne.s   Waitmouse       ; se no ritorna a waitmouse e ripeti
6                               ; (il .s è un equivalente del .b per questo
7                               ; tipo di comandi: bne.s = bne.b)
8       clr.b   dato1           ; AZZERA DATO1
9       rts                     ; esci
10
11 dato1:
12      dc.b    $30     ; dc.b significa METTI IN MEMORIA IL SEGUENTE BYTE
13                      ; in questo caso viene messo un $30 sotto dato1:
```

Prima di uscire con l'RTS si azzererebbe il byte contrassegnato con la label dato1:, che sarebbe allocato nella fase di assemblaggio (o compilazione) a qualche indirizzo assoluto ben preciso, ad esempio se il programma fosse assemblato dall'ASMONE a partire da *$50000*, si troverebbe in memoria dopo l'assemblaggio un CLR.B $5001c, cioè l'indirizzo reale di dato1:, non certo CLR.B DATO1, essendo dato1: un nome dato dal programmatore per contrassegnare il dc.b $30; di qui si intuisce anche l'utilità delle label, infatti se si dovesse scrivere il listato indicando l'indirizzo numerico tutte le volte, nel caso che si aggiungesse una routine nel mezzo del programma si dovrebbero riscrivere tutti gli indirizzi. Per vedere a quale indirizzo vengono assemblate le label, basta usare il comando *<D>* dell'ASMONE: ad esempio dopo aver assemblato LEZIONE1a.s facendo *<D waitmouse>* otterrete il disassemblato della memoria a partire da waitmouse, e nel listato non appariranno le label, ma gli indirizzi reali.

Nei sorgenti esempio del corso noterete che non vengono mai indicati indirizzi numerici, ma solo LABEL, a parte gli indirizzi speciali come *$dffxxx* o *$bfexxx*. Nell'ultimo esempio ho usato un dc.b, che è un comando dell'assemblatore che serve per inserire bytes definiti; per esempio per inserire un *$12345678* in un dato punto del programma, dovrò usare il comando DC, e lo posso usare nelle 3 forme .B (BYTE), .W (WORD) e .L (LONGWORD):

```
1       dc.b    $12,$34,$56,$78 ; in bytes
2
3       dc.w    $1234,$5678     ; in words
4
5       dc.l    $12345678       ; in longwords
```

Questo comando si usa anche per mettere delle frasi in memoria, ad esempio per mettere nel listato il testo che dovrà essere stampato a video da una routine di PRINT che stampi ciò che è alla label TESTO:

```
1  TESTO:
2       dc.b    "tanti saluti"
3
4  oppure:
5       dc.b    'tanti salutì
6
7  Di solito si termina il testo con uno zero:
8
9       dc.b    "tanti saluti",0
```

Bisogna ricordarsi di mettere il testo tra virgolette e di usare il dc.b, non il dc.w o il dc.l!! I caratteri sono lunghi un byte ciascuno, e corrispondono ad un certo byte: ad esempio provate a fare *<?"a">* , e vedrete che corrisponde a `$61`, quindi scrivere `dc.b "a"` sarà equivalente a scrivere `dc.b $61`. Attenzione che le lettere grandi hanno valore diverso! un `"A"` per esempio è `$41`. L'uso più comune del `dc.b` è quello di definire byte, word, o zone più grandi dove verranno tenuti dei dati, ad esempio se si volesse fare un programma che registri il numero di volte che si preme un certo tasto, bisognerà definire una label seguita per esempio da un byte azzerato, ed ogni volta si aggiungerà 1 con il comando ADD a quella label, ossia a quel byte sotto la label, e all'uscita basterà leggere il valore del byte:

```
        ; se il tasto è premuto allora ADDQ.B #1,NUMPREMUTO, ossia
        ; aggiungi uno al byte sotto la label numpremuto.

NUMPREMUTO:
        dc.b    0
```

All'uscita del programma lo 0 iniziale sarà cambiato nel numero di volte che il tasto è stato premuto. Un esempio simile è in `LEZIONE2a.s`, che contiene anche un ampio commento. Vi consiglio di caricarlo in un altro buffer di testo: per selezionare uno dei 10 disponibili basta premere un tasto da *<F1>* a *<F10>*, se per esempio avete `LEZIONE2.TXT` nel buffer di *<F1>*, premete *<F2>*, e caricateci `LEZIONE2a.s` con il comando *<R>*. In seguito potete caricare `LEZIONE2b.s` e i seguenti nel buffer di *<F3>*, *<F4>*..., in modo da averli sempre a disposizione premendo un solo tasto; è meglio comunque se seguite la `LEZIONE.TXT`, e man mano che trovate l'indicazione del sorgente esempio, continuate caricandolo in un altro buffer, eseguendolo e verificandolo, dopodiché ritornate a leggere la LEZIONE da dove eravate: questo credo sia il miglior sistema per imparare, infatti si fa un po di teoria e si verifica subito.

Avete letto i commenti di `LEZIONE2a.S`?

Avrete visto l'importanza che hanno il byte, la word e la longword: per quanto riguarda il binario, per contare i bit, si comincia da destra e si va verso sinistra, al "contrario" insomma, e si parte da 0, non da 1, dunque un byte (che ha 8 bit) parte 0 e va fino al 7. Per esempio in questo numero:

```
%000100010000
```

Sono "accesi" i bit 4 e 8. Per aiutarvi a numerarli potete fare così:

```
                ;5432109876543210       — un utilizzo intelligente del ;
    move.w    #%0011000000100100,$dffxxx
```

In questo caso sono "accesi" i bit 2, 5, 12 e 13 della WORD. Ricordo che un byte ha 8 bit, una word ne ha 16 (da 0 a 15), una longword ne ha 32 (da 0 a 31). Nell'istruzione

```
BTST #6,$bfe001
```

Si controlla se il bit 6 del byte `$bfe001` è azzerato. Se fosse:

```
;76543210
%01000000
```

Il bit 6 è invece ad 1, dunque il mouse non è premuto!!!

Per ricapitolare, un *byte* è fatto di 8 bit: per indicarli, il primo a destra è il bit 0, detto anche *bit meno significativo*. La numerazione procede da destra verso sinistra fino al 7, (ossia l'ottavo perché si parte da 0 anziché da 1: 01234567, ossia 8 bit); il bit 7 è detto *bit più significativo*. È più significativo perché conta di più allo stesso modo in cui conta di più, nel bigliettone da centomila, l'uno più a sinistra degli zeri più a destra. Un byte, al massimo può valere 255, ossia %11111111.

Una WORD invece è fatto di 16 bit, ovvero due byte, allo stesso modo si parte da destra col bit 0, sempre il meno significativo, fino al bit 15 ultimo a sinistra, il più significativo. Al massimo può contenere 65535.

Una LongWord è fatta di 32 bit, da 0 a 31, ossia 4 bytes, o 2 word, o se preferite una word e 2 bytes, insomma sempre 32 bit attaccati l'uno all'altro che al massimo possono contenere 4294967299 (4 miliardi!! come la lotteria!).

Ora procederemo con diversi modi di indirizzare: abbiamo visto che se facciamo ad esempio un `CLR.W $100`, azzereremo le locazioni *$100* e *$101*, ossia una word a partire da *$100* (essendo una word 2 bytes, e le locazioni divise in bytes, puliremo 2 bytes!!). Allo stesso modo un `MOVE.B $100,$200` copierà il contenuto di *$100* in *$200*. Questo si può indicare anche con le LABEL invece di specificare l'indirizzo, ad esempio `MOVE.B LABEL1,LABEL2`, ovvero copia il byte di *LABEL1* in *LABEL2*. Ci sono però anche diversi modi di indirizzare, infatti posso fare un `MOVE.L #$50000,LABEL2`, ossia mettere un valore *fisso* in *LABEL2*. Se ad esempio *LABEL2* fosse all'indirizzo *$60000*, muoveremmo il valore `$00050000` in *$60000*, ovvero i bytes facendo un *<M $60000>*: `00 05 00 00`. Infatti quando c'è il simbolo del cancelletto (`#`) prima di un numero o di una label significa che si sta muovendo un valore stabilito, e non il valore contenuto nell'indirizzo indicato con quel valore, come avviene se non ci sono cancelletti prima del numero o della LABEL. Per esempio analizziamo questi 2 casi:

```
1)      MOVE.L  $50000,$60000   ; il valore contenuto negli indirizzi
                                ; di memoria $50000,$50001,$50002,$50003
                                ; vengono copiati in $60000,
                                ; $60001,$60002,$60003, ossia una longword
                                ; composta di 4 bytes viene copiata da un
                                ; indirizzo all'altro.

2)      MOVE.L  #$50000,$60000  ; Questa volta in $60000 viene messo il
                                ; numero indicato dopo il cancelletto,
                                ; ossia $50000. Da notare che questa
                                ; volta l'indirizzo $50000 non viene letto
                                ; e non c'entra assolutamente, viene
                                ; implicato solo il $60000.
```

Se si usano delle label, non ci sono cambiamenti:

```
1)      MOVE.L  CANE,GATTO      ; Il contenuto della longword cane, ossia
                                ; $00123456 viene copiato nella longword
                                ; GATTO (infatti $123456 è la prima cosa
                                ; sotto la label CANE)
```

prima dell'istruzione:

```
CANE:
        dc.l    $123456

GATTO:
        dc.l    0
```

dopo l'istruzione:

```
CANE:
        dc.l    $123456

GATTO:
        dc.l    $123456

2)      MOVE.L  #CANE,GATTO     ; Questa volta l'INDIRIZZO della label
                                ; CANE viene copiato nella label GATTO
```

prima dell'istruzione:

```
                        ; SUPPONIAMO CHE LA LABEL CANE: sia alla locazione
                        ; $34500, cioè facendo un M CANE dopo aver assemblato
                        ; compare un :
                        ; 00034500 00 12 34 56 00 00 00 00 .....
                        ;          (cane)      (gatto)

CANE:
        dc.l    $123456

```

```
10 GATTO:
11         dc.l    0
```

dopo l'istruzione:

```
1  CANE:
2          dc.l    $123456
3
4  GATTO:
5          dc.l    $34500  ; ossia DOVE è LA LABEL CANE IN MEMORIA.
```

Da notare che se si faceva un `MOVE.W #CANE,GATTO` o un `MOVE.B #CANE,GATTO` l'assemblatore avrebbe dato un errore, in quanto un *indirizzo è lungo una longword*. In memoria un `MOVE.L #LABEL,LABEL` si trasforma in un istruzione del tipo `MOVE.L #$12345,$12345`, ossia l'assemblatore scrive l'indirizzo reale al posto delle label. Questo lo potete verificare con `LEZIONE2b.s`.

Ora affronteremo gli altri indirizzamenti con i registri (che sono più difficili); come avevo già accennato, ci sono 8 registri dati e 8 registri indirizzi: ossia `D0,D1,D2,D3,D4,D5,D6,D7` sono i registri *dati*, mentre `a0,a1,a2,a3,a4,a5,a6,a7` sono i registri *indirizzi*. Premetto che il registro `A7` è detto anche `SP` o *stack pointer*, ed è un registro particolare di cui parleremo dopo, quindi considerate di usare i registri indirizzi solo fino ad `a6`. questi indirizzi sono lunghi una longword ciascuno, ed sono in pratica una piccola memoria dentro il 68000, che di conseguenza è molto veloce. Tramite i registri si possono fare varie cose, infatti esiste una sintassi particolare per i registri. Innanzitutto non si può lavorare per byte con i registri INDIRIZZI: per esempio un `move.b LABEL,a0` da un messaggio di errore. Con i registri indirizzi `a0`, `a1`, etc... si può dunque lavorare per longword o per word. Con i registri Dati `D0`, `D1`, etc..., invece si possono usare sia .b che .w che .l. I registri indirizzi sono dedicati a contenere indirizzi, ed hanno comandi dedicati, come il `LEA`, che significa LOAD ENTIRE ADDRESS, ovvero carica l'indirizzo interamente nel registro (infatti questo comando non può essere `lea.b`, `lea.w` o `lea.l`, ma solo `LEA` essendo sempre .L) Per esempio, per mettere un valore nei registri indirizzi si possono usare 2 metodi:

```
1  1)      MOVE.L  #$50000,A0      (oppure MOVE.L #LABEL,a0)
2
3  2)      LEA     $50000,a0       (oppure LEA LABEL,A0)
```

Mentre il primo metodo si può usare sia con gli indirizzi che con i registri (es: `move.l #$50000,d0` - `move.l #$50000,LABEL` - `MOVE.L #$LABEL,LABEL...`)

P.S: scrivere `move.l #$50000,d0` o `MOVE.L #$50000,D0` è identico, si può scrivere anche `MoVe.L #$50000,d0`, il risultato a livello di programma è identico, solo che esteticamente potete creare delle situazioni simpatiche o orribili. Va fatto un discorso diverso per le LABEL: le label possono essere identificate anche se in un punto del listato le scrivete in minuscolo e in un altro in maiuscolo, questo però solo perchè è settata nelle preferenze del TRASH'M-ONE questa opzione, che è "UCase=UCase" nel menù "Assembler/Assemble..", che significa "Upper Case=Lower Case, ossia lettere grandi=lettere piccole". Se togliete questa opzione, nel riconoscimento delle label sarà tenuto presente anche il maiuscolo/minuscolo, per cui Cane: sarà diverso da CANE: o da cAne: o da caNe, eccetera eccetera.

il secondo metodo con il `LEA` si può usare solo con i registri indirizzi, di conseguenza si intuisce che questo modo è più veloce: ricordatevi quindi che se volete mettere un indirizzo in un registro `a0`, `a1`... dovete usare il LEA seguito dall'indirizzo SENZA CANCELLETTO e dal registro in questione. Fate attenzione a questi 2 esempi:

```
1  1)      MOVE.L  $50000,a0       ; metti in a0 il valore contenuto nella
2                                  ; locazione $50000 (+$50001,$50002 e $50003
3                                  ; in quanto 1 locazione è lunga 1 byte, ed
4                                  ; il move.l copia 4 bytes = 4 locazioni a
5                                  ; partire in questo caso da $50000
6
7  2)      LEA     $50000,a0       ; metti il numero $50000 in a0
```

State attenti quindi a maneggiare i MOVE con o senza il cancelletto ed i LEA, perché è facile nei primi tempi sbagliarsi e mettere l'indirizzo invece del valore di quell'indirizzo nel registro o viceversa. Come ulteriore commento di questa differenza consultate il programmino esempio LEZIONE2c.s

Con i registri indirizzi sono possibili vari tipi di indirizzamento: Per cominciare analizziamo queste 2 istruzioni:

```
1       move.l  a0,d0    ; Metti il numero contenuto in a0 nel registro d0
2       move.l  (a0),d0  ; Metti la longword contenuta dall'indirizzo in a0
3                        ; nel registro d0
```

L'indirizzamento tra parentesi si dice *indiretto*, perché anziché venir copiato *direttamente* il valore in a0 viene copiato il valore contenuto nell'indirizzo che è in a0. Un esempio pratico è in LEZIONE2d.s

Usando l'indirizzamento indiretto si può agire sugli indirizzi *indirettamente*, ad esempio mettendo l'indirizzo del tasto del mouse e del colore 1 nei registri si può riscrivere il listato della lezione 1. Così ho fatto in LEZIONE2e.s

Facciamo gli ultimi esempi per togliere gli eventuali dubbi sull'indirizzamento indiretto:

```
1       move.l a0,d0            ; copia il valore di A0 nel reg. d0
2       move.b (a0),d0          ; copia il byte contenuto nell'indirizzo
3                               ; in a0 nel reg. d0
4       move.w (a0),(a1)        ; copia la word contenuta dall'indirizzo
5                               ; in a0 all'indirizzo contenuto in a1
6                               ; (e seguente, essendo una word fatta di
7                               ; 2 bytes, ossia 2 indirizzi!)
8       clr.w   (a3)            ; pulisce (azzera) la word (2 bytes) "dentro"
9                               ; l'indirizzo in a3 - Più precisamente,
10                              ; viene azzerato il byte  dell'indirizzo in
11                              ; a3 e l'indirizzo seguente.
12      clr.l   (a3)            ; Come sopra, ma sono azzerati 4 indirizzi
13                              ; (una long = 4 bytes = 4 indirizzi)
14      move.l  d0,(a5)         ; viene copiato il valore di d0 nell'indirizzo
15                              ; contenuto in a5 ( più precisamente dovrei
16                              ; dire nell'indirizzo in a5, e nei 3 seguenti,
17                              ; in quanto una long occupa 4 indirizzi)
18      move.l  d0,a5           ; viene copiato il valore di d0 in a5
```

Mi raccomando! Toglietevi ogni dubbio sugli indirizzamenti fin qui studiati, consultando anche i sorgenti fino a LEZIONE2e.s, perché gli indirizzamenti di cui parlerò ora si basano su quelli indiretti normali.

Vi comunico che questa è la parte più astratta della lezione2, in quanto si devono imparare gli ultimi indirizzamenti del processore, ma vi assicuro che già dalla lezione 3 metterete in pratica il tutto e visualizzerete degli effetti video col copper!, quindi considerate che passata questa parte il resto del corso sarà tutto più PRATICO: a ogni spiegazione corrisponderà un nuovo effetto speciale o colore ultravivace, dunque fate lo sforzo di non annoiarvi e di non lasciar perdere ora, perché io stesso lasciai perdere all'incirca a questo punto la prima volta che tentai di imparare a programmare in ASM, proprio perché ero scoraggiato dal CASINO di comandi e parentesi aperte e chiuse che poi non riuscivo più a seguire. Vi assicuro però che una volta imparato a leggere i comandi, potete partire come una fucilata e imparare da voi leggendo listati qua e là, facendo passi sempre più grandi: è come imparare le regole di uno sport: uno che non conosce il set di istruzioni del 68000 è come uno che non conosce le regole, ad esempio, del calcio: guardando le partite (i listati) costui non capirà nulla di cosa stanno facendo quegli scalmanati in un campo a dare calci ad una palla, e si annoierà a morte, ma una volta capite le regole (indirizzamenti) potrà interpretare le fasi delle partite ed imparare sempre di più le tecniche di gioco (i trucchi della programmazione ed i registri della grafica).

Vediamo altri 2 modi di indirizzamento:

```
1       move.l  (a0)+,d0        ; Indiretto con post-incremento
2       move.l  -(a0),d0        ; Indiretto con pre-decremento
```

Analizziamo il primo indirizzamento ipotizzando questa situazione:

```
        lea     NONNO,a0        ; mettiamo in a0 l'indirizzo di NONNO:
        MOVE.L  (a0)+,d0        ; mettiamo in d0 il valore .L contenuto
                                ; dall'indirizzo in a0, ossia $3231020
                                ; (come un normale MOVE.L (a0),d0)
                                ; dopodichè AGGIUNGIAMO 4 AL VALORE IN a0
                                ; ovvero andiamo a PUNTARE alla long seguente
                                ; con l'indirizzo in a0; se fosse stato un
                                ; move.w (a0)+,d0 ad a0 DOPO (POST-INCREMENTO)
                                ; sarebbe stato aggiunto 2 (una word=2),
                                ; mentre nel caso di un MOVE.B (a0)+,d0
                                ; sarebbe stato aggiunto 1, (un byte),
                                ; ovvero sarebbe andato a puntare l'indirizzo
                                ; seguente.
        MOVE.L  (a0)+,d1        ; stessa cosa: copia in d1 il valore .L
                                ; contenuto nell'indirizzo in a0, che ora
                                ; contiene l'indirizzo di NONNO+una longword,
                                ; ovvero NONNO+4, ossia $13478.
        rts                     ; ESCE!

NONNO:
        dc.l    $3231020,$13478

        END
```

Possiamo tradurre questo tipo di indirizzamento con un 2 istruzioni:

```
1)      MOVE.L  (a0)+,LABEL
```

è equivalente a:

```
1b)     MOVE.L  (A0),LABEL      ; copia una long dall'indirizzo in a0
                                ; nella label
        ADDQ.W  #4,a0           ; Aggiungi 4 ad a0 (.L=4)
                                ; NOTA: se si addiziona un numero minore di
                                ; 9 si usa il comando ADDQ invece di ADD
                                ; perchè è dedicato a tali numeri e veloce.
                                ; Inoltre su registri INDIRIZZI se il numero
                                ; che aggiungiamo o sottraiamo è minore di
                                ; $FFFF, ossia una word, si può usare il .W
                                ; anzichè il .L, e si agirà comunque su
                                ; tutta la longword dell'indirizzo.
```

Allo stesso modo:

```
2)      MOVE.W  (a0)+,LABEL
```

è equivalente a:

```
2b)     MOVE.W  (A0),LABEL      ; copia una word dall'indirizzo in a0
                                ; nella label
        ADDQ.W  #2,a0           ; Aggiungi 2 ad a0 (.W=2)
```

Allo stesso modo:

```
3)      MOVE.B  (a0)+,LABEL
```

è equivalente a:

```
3b)     MOVE.B  (A0),LABEL      ; copia il byte contenuto nell'indirizzo
                                ; in a0 nella label
        ADDQ.W  #1,a0           ; Aggiungi 1 ad a0 (.B=1)
```

Dunque, riassumendo in altri termini, l'indirizzamento indiretto con post incremento si può paragonare ad un operaio di una catena di montaggio che *prima* esegue il suo `MOVE` o la sua istruzione sul pezzo che sta sul nastro trasportatore, e ogni volta che ha fatto il suo lavoro sul pezzo sposta *avanti* il nastro trasportatore (l'indirizzo in *a0*) con un pedale (il +). Un esempio di loop può risultare più chiaro:

```
Inizio:
        lea     $60000,a0       ; inizio pulizia
        lea     $62000,a1       ; fine pulizia
CLELOOP:
        clr.l   (a0)+   ; azzera una long dall'indirizzo in A0 e aumenta a0
                        ; di una long, ossia di 4 indirizzi, in altre
                        ; parole pulisci una long e vai alla prossima
        cmp.l   a0,a1   ; A0 è arrivato a $62000? Ossia, a0 è uguale ad a1?
        bne.s   CLELOOP ; se non ancora, continua con un altro ciclo CLELOOP
        rts
```

Come si vede, questo programmino pulisce la memoria dall'indirizzo *$60000* a *$62000*, utilizzando un `clr (a0)+` ripetuto fino a che non si è arrivati all'indirizzo desiderato. Un esempio simile lo potete trovare in `Lezione2f.s`

Ora impareremo l'indirizzamento indiretto con pre-decremento, ossia un indirizzamento opposto a quello appena descritto, infatti invece di aumentare l'indirizzo contenuto nel registro dopo aver eseguito l'operazione, con un `clr.l -(a0)`, per esempio, prima viene decrementato a0, poi viene eseguita l'istruzione sul nuovo indirizzo (in questo caso *a0-4*). Esempio:

```
        lea     NONNO,a0        ; mettiamo in a0 l'indirizzo di NONNO:
        MOVE.L  -(a0),d0        ; a0 viene decrementato, in questo caso
        rts                     ; essendo un istruzione .L viene decrementato
                                ; di 4, dopodichè viene copiato in d0
                                ; il valore .L contenuto dall'indirizzo
                                ; in a0, ossia $12345678, cioè NONNO-4
                                ; (nel registro rimane il valore iniziale-4)
        dc.l    $12345678       ; se fosse stato un
NONNO:                          ; move.w -(a0),d0 ad a0 PRIMA (PRE-INCREMENTO)
        dc.l    $ffff0f0f       ; sarebbe stato sottratto 2 (una word=2),
                                ; mentre nel caso di un MOVE.B -(a0),d0
        END                     ; sarebbe stato sottratto 1, (un byte),
                                ; ovvero sarebbe andato a puntare l'indirizzo
                                ; precedente.
```

Possiamo tradurre questo tipo di indirizzamento con un 2 istruzioni:

```
1)      MOVE.L  -(a0),LABEL
```

è equivalente a:

```
1b)     SUBQ.W  #4,a0           ; Sottrai 4 ad a0 (.L=4)
                                ; NOTA: se si sottrae un numero minore di
                                ; 9 si usa il comando SUBQ invece di SUB
                                ; perchè è dedicato a tali numeri e veloce.

        MOVE.L  (A0),LABEL      ; copia una long dall'indirizzo in a0
                                ; nella label
```

Allo stesso modo:

```
2)      MOVE.W  -(a0),LABEL
```

è equivalente a:

```
2b)     SUBQ.W  #2,a0           ; Sottrai 2 ad a0 (.W=2)
        MOVE.W  (A0),LABEL      ; copia una word dall'indirizzo in a0
                                ; nella label
```

Allo stesso modo:

```
3)      MOVE.B  -(a0),LABEL
```

è equivalente a:

```
3b)     SUBQ.W  #1,a0           ; sottrai 1 ad a0 (.B=1)
        MOVE.B  (A0),LABEL      ; copia il byte contenuto nell'indirizzo
                                ; in a0 nella label
```

Riassumendo con l'operaio come prima, l'indirizzamento indiretto con pre decremento si può paragonare sempre ad un operaio di una catena di montaggio che *prima* sposta *indietro* il nastro

trasportatore (l'indirizzo in a0) con un pedale (il -), *poi* esegue il suo MOVE o la sua istruzione sul pezzo che sta sul nastro trasportatore. Un esempio di loop:

```
Inizio:
        lea     $62000,a0       ; inizio pulizia
        lea     $60000,a1       ; fine pulizia
CLELOOP:
        clr.l   -(a0)   ; diminuisci a0 di una long e azzera quella long
                        ; in altre parole vai alla precedente long e puliscila
        cmp.l   a0,a1   ; A0 è arrivato a $60000? Ossia, a0 è uguale ad a1?
        bne.s   CLELOOP ; se non ancora, continua con un altro ciclo CLELOOP
        rts
```

Come si vede, questo programmino pulisce la memoria dall'indirizzo *$62000* a *$60000*, utilizzando un `clr -(a0)` ripetuto fino a che non si è arrivati all'indirizzo desiderato (All'indietro però! mentre con `(a0)+` si parte da *$60000* e di 4 in 4 si arriva a *$62000*, in questo caso si parte da *$62000* e si arriva a *$60000* indietreggiando di 4 in 4). Vedete `Lezione2g.s` e `Lezione2h.s` per verificare gli ultimi 2 indirizzamenti.

Ora impareremo come usare la distanza di indirizzamento: un `MOVE.L $100(a0),d0` copia in `d0` la long contenuta dall'indirizzo in *a0+$100*, ossia: se per esempio in `A0` avevamo l'indirizzo *$60200*, in `d0` ci andrà la longword contenuta dall'indirizzo *$60300*. Allo stesso modo un `MOVE.L -$100(a0),d0` copierà in `d0` la long a partire dall'indirizzo *$60100*. Da notare che `a0` non cambia di valore: semplicemente il processore calcola ogni volta a che indirizzo operare, facendo la somma tra il valore prima della parentesi e l'indirizzo nel registro tra parentesi. La massima distanza di indirizzamento è da -32768 a 32767 (*-$7FFF*, *$8000*) Un esempio su questo tipo di indirizzamento è `Lezione2i.s`

L'ultimo tipo di indirizzamento è questo:

```
        MOVE.L  50(a0,d0),label
```

che ha sia una *distanza di indirizzamento* (il 50) che un *indice* (il `d0`): la distanza di indirizzamento e il contenuto di `d0` sono tutti sommati per definire l'indirizzo da cui copiare il contenuto. In pratica è come la distanza di indirizzamento, ma in più viene aggiunto anche il contenuto dell'altro registro alla distanza di indirizzamento, che in questo caso però va da un minimo di -128 ad un massimo di +128. Non vi voglio annoiare con altri esempi su questo indirizzamento, potrete verificarlo quando lo troverete nei prossimi listati.

Per terminare la LEZIONE2, che se avete seguito bene vi rende in grado di seguire le operazioni di un qualsiasi programma in ASM, è indispensabile spiegare il ciclo `DBRA`, che è usato moltissimo: usando un registro dati si possono far eseguire delle istruzioni varie volte, basta mettere nel registro dati (sia esso `d0`, `d1...`) il numero di volte-1. Ad esempio la routine che pulisce la memoria fatta con il `CLR.l (a0)+` può essere modificata con un loop `DBRA` che faccia eseguire la pulizia il numero desiderato di volte:

```
Inizio:
        lea     $60000,a0       ; Inizio
        move.l  #($2000/4)-1,d0 ; Metti in d0 il numero di cicli necessari
                                ; per cancellare $2000 bytes: cioè
                                ; $2000/4 (ovvero DIVISO 4, perchè ogni
                                ; clr.l pulisce 4 bytes), il tutto -1,
                                ; perchè il loop viene eseguito una volta
                                ; in più.
CLEARLOOP:
        CLR.L   (a0)+
        DBRA    d0,CLEARLOOP
        rts
```

Questa routine pulisce da *$60000* a *$62000* come l'esempio precedente in cui con il comando `CMP` si COMPARA `a0` con `a1`, ossia si verifica se siamo arrivati a *$62000* che è in `a1`. In questo caso invece viene eseguito il `CLR` 2047 volte, provate infatti a fare *<?($2000/4)-1>* da ASMONE. Il `DBRA` funziona in questo modo: se per esempio la prima volta in d0 viene messo 2047, viene

eseguito il CLR, poi arrivati al DBRA d0 viene diminuito di 1 e il processore salta nuovamente al CLR, lo esegue eccetera, fino a che d0 è esaurito. Bisogna mettere il numero di cicli necessari meno uno perché la prima volta il ciclo viene eseguito senza decrementare d0.

Come ultimo esempio studiatevi Lezione2l.s, che ha delle subroutines richiamate con il BSR e il ciclo DBRA in azione, utile per capire la struttura di un programma complesso.

Per terminare vorrei farvi notare la differenza tra un BSR ed un BEQ/BNE: nel caso del BSR label, il processore salta ad eseguire la routine sotto la label, fino a che non trova l'RTS, che lo fa tornare ad eseguire l'istruzione sotto il BSR label, dunque si può dire che ha eseguito una *sottoroutine*, cioè una routine eseguita in mezzo ad un'altra routine:

```
principale:
        move.l  roba1,d0

        move.l  roba2,d1

        bsr.s   sottoposto

        move.l  roba3,d2

        move.l  roba4,d3

        rts     ; FINE DELLA ROUTINE PRINCIPALE, TORNA ALL'ASMONE


sottoposto:
        move.l  robaccia,d4

        move.l  robaccia2,d5

        rts     ; FINE DELLA SOTTOROUTINE, TORNA A  "move.l roba3,d0", ossia
                ; sotto il bsr.s sottoposto
```

Nel caso di una *diramazione* beq/bne invece si prende O una strada O l'altra:

```
principale:
        move.l  roba1,d0

        move.l  roba2,a0

        cmp.b   d0,a0
        bne.s   strada2

        move.l  roba3,d1

        cmp.b   d1,a0
        beq.s   strada3

        move.l  roba4,d0

        rts     ; FINE DELLA ROUTINE PRINCIPALE, TORNA ALL'ASMONE


strada2:
        move.l  robaccia,d5

        move.l  robaccia2,d6

        rts     ; FINE ROUTINE, TORNA ALL'ASMONE, non sotto il bne!!!
                ; qua abbiamo scelto questa strada, e come si trova un RTS
                ; si torna all'ASMONE!!!


strada3:
        move.l  robaccia3,d1

        move.l  robaccia4,d2

        rts     ; FINE ROUTINE, TORNA ALL'ASMONE, non sotto il beq!!!
                ; qua abbiamo scelto questa strada, e come si trova un RTS
                ; si torna all'ASMONE!!!
```

Lo stesso vale per il `BRA label`, che significa SALTA A label, equivalente del `JMP`, per cui è come un treno che trova uno scambio ai binari, non torna allo scambio quando ha finito il binario!! Arriva alla fine del binario e basta, senza teletrasporti alla star trek all'indietro.

Per un'ultima precisazione sui registri indirizzi, vedetevi `Lezione2m.s`

Per caricare la `LEZIONE3.TXT` potete fare in due modi: o scrivete *<R>* e andate a capo, facendo aprire il requester dove potete selezionare col mouse quale testo caricare (in questo caso *<df0:SORGENTI/LEZIONE3.TXT>*), oppure dovete assicurarvi di trovarvi nella directory giusta con un *<V df0:LEZIONI>* e potete caricarla in seguito con un semplice *<R LEZIONE3.TXT>*

CAPITOLO 3

LEZIONE 3 - LA PRIMA COPPERLIST

Ora procederemo nella pratica, ma prima vi consiglio di caricarvi in un buffer di testo il file `68000.TXT` che è un riassunto della lezione2. Questo potrà esservi utile nel caso che non ricordaste un indirizzamento o una istruzione leggendo i listati di questa lezione, che presuppongono la familiarità con gli indirizzamenti e le istruzioni affrontate prima. Il quel testo sono spiegati tutti gli indirizzamenti, anche quelli che non sono quasi mai usati, dunque leggetelo ma non preoccupatevi se non capite gli indirizzamenti con *indice*, tanto nella lezione3 non saranno usati!

In questa lezione si comincia a visualizzare qualcosa sullo schermo: per fare questo dobbiamo scrivere una **CopperList**, cioè un programma per il chip *Copper* che si occupa della grafica, che abbiamo già usato per cambiare di colore lo schermo (`$dff180` è un registro del copper, che si chiama `COLOR00`). Per ora però abbiamo solo fatto delle modifiche col processore direttamente nei registri, e come avete potuto notare eseguendo i listati con *<AD>* un istruzione alla volta, quando mettiamo un valore nel `COLOR00` (ossia *$dff180*) avviene solo un brevissimo lampo, e subito torna il colore normale del sistema operativo, ossia dell'ASMONE. Solo facendo un ciclo in cui si immette continuamente un numero si può colorare tutto lo schermo, ma una volta usciti dal programmino il colore ritorna inesorabilmente quello normale. Questo avviene perché lo schermo che vediamo con finestre, scritte e tutto il resto è il frutto di una *CopperList*, e precisamente di una *CopperList di sistema*. La copperlist non è altro che una specie di:

```
1       MOVE.W  #$123,$dff180   ; COLOR00 - immetti il colore 0
2       MOVE.W  #$123,$dff182   ; COLOR01 - immetti il colore 1
3       eccetera...
```

Che viene eseguito continuamente, quindi ecco spiegato perché se col processore cambiamo il colore subito torna quello di sistema: perché la copperlist ridefinisce ogni cinquantesimo di secondo tutti i colori!!!! Intuirete che per visualizzarsi delle figure in pace non è possibile continuare a fare loop tentando di combattere con la copperlist di sistema che ridefinisce simultaneamente tutto, ma dovremo togliere di mezzo la copperlist di sistema e farcene una tutta nostra. *Niente di più facile!* Come ho già premesso, la copperlist non è altro che una sfilza di `MOVE` che mettono dei valori nei registri del COPPER, ossia quelli `$dffxxx`; comunque non sono dei `move` fatti col processore, ma fatti dal copper stesso, che, non a caso, esegue questa COPPERLIST

indipendentemente mentre col processore stiamo facendo altre cose... questo è uno dei motivi per cui su PC non hanno LIONHEARTH o PROJECT X dell'Amiga.

Dovremo quindi scrivergli proprio un *listato*, come facciamo per il 68000, dopodiché dovremo informare il COPPER dove si trova il nostro per farglielo leggere ed eseguire al posto di quello del WorkBench. Il copper ha **solo** 3 istruzioni, di cui in pratica ne vengono usate solo 2: le due usate sono il `MOVE` ed il `WAIT`; quella che non usa nessuno è lo `SKIP`, quindi di quella ne parleremo solo se la troveremo in un listato di esempio. Il `MOVE` è *facilissimo*! Avete presente un:

```
1       MOVE.W  #$123,$dff180   ; Immetti il colore RGB nel COLOR00
```

Si traduce in copperlist in:

```
1       dc.w    $180,$123       ; si mettono in memoria direttamente i
2                               ; numeri col dc.w, tanto basta
3                               ; imparci 2 istruzioni solamente!
```

Ossia: si deve mettere prima l'indirizzo di destinazione, senza il *$dff* come abbiamo già visto quando mettiamo `$dff000` in `a0`, basta fare `$180(a0)`: allo stesso modo i progettisti hanno pensato bene di risparmiarci la fatica di fare `$DFF` tutte le volte e così basta mettere il `$180`, o il `$182` o qualsiasi altro registro del COPPER, infatti SOLO i registri del Copper possono essere scritti dalla COPPERLIST, e si può accedere solo ai registri **pari**, come `$180`, `$182`... mai `$181`, `$183`!!!!, inoltre potete modificare solo una WORD alla volta. Come avete visto, la COPPERLIST non viene assemblata come i comandi del 68000 che vengono trasformati da *istruzioni* come `RTS`, `MOVE`... a `$4e75`, etc., bensì si devono mettere i *bytes* come sono realmente in memoria e come sono letti dal coprocessore COPPER: per la COPPERLIST appunto dobbiamo usare il comando `DC` per metterla in memoria a forza di BYTES, ma è facilissimo. Per esempio per definire i primi 4 colori:

```
1   COPPERLIST:
2       dc.w    $180,$000       ; COLORE 0 = NERO
3       dc.w    $182,$f00       ; COLORE 1 = ROSSO
4       dc.w    $184,$0F0       ; COLORE 3 = VERDE
5       dc.w    $186,$00F       ; COLORE 4 = BLU
```

Vi ricordate come è il formato dei colori? RGB=RED, GREEN, BLU. Per avere un aiuto in ogni momento sul significato dei registri `$dffXXX` fate *<=C 180>* oppure *<=C numero>* e avrete un breve riassunto (in inglese). Per esempio fate *<=c 006>* e vedrete il nome e la spiegazione del registro che avete usato per far lampeggiare il colore. Per vedere tutti i registri fate semplicemente un *<=C>*.

Il `WAIT` invece serve per aspettare una certa linea dello schermo, ad esempio se si vuol fare il colore di sfondo (`color0`) nero fino a metà, mentre nella metà inferiore si vuole blu, basta mettere un

```
1       dc.w    $180,0          ; colore 0 NERO
```

seguito da un `WAIT` che aspetta la metà dello schermo, dopodiché

```
1       dc.w    $180,$00F       ; colore 0 BLU
```

Con questo stratagemma si può cambiare l'intera palette (i colori) a qualsiasi linea del video, cosa che invece su PC in VGA nemmeno si sognano, infatti anche se i giochi Amiga solitamente hanno schermi di soli 32 colori, cambiando la tavolozza dei colori ogni tanto man mano che lo schermo scende si possono fare più tonalità di una VGA a 256 colori, specialmente se si considera che con un solo colore di sfondo si può fare una sfumatura cambiando il colore ad ogni linea, come faremo nel primo listato di questa lezione. Il comando `WAIT` si presenta in questa forma:

```
1       dc.w    $1007,$FFFE     ; WAIT coordinata X= $10, Y= $07
```

Questo comando significa: ATTENDI LA LINEA ORIZZONTALE $10, colonna 7 (cioè al settimo punto a partire da sinistra; i punti sono detti **pixel**). Il `$FFFE` significa `WAIT`, e va messo comunque

tutte le volte, mentre il primo byte è la linea orizzontale (x) da aspettare e il secondo è quella verticale (y).

Lo schermo infatti è fatto di molti punti disposti l'uno accanto all'altro, come un foglio a quadretti molto piccoli, ad esempio la carta millimetrata. Per indicare il punto (pixel) situato (come nella battaglia navale) alla posizione 16,7, ossia a 16 punti dal bordo superiore del foglio verso il basso e 7 dal bordo sinistro verso destra, indicherò `$1007`. ($10=16!). Di solito basta indicare la linea orizzontale al suo inizio, (la posizione è `$07` anziché `$01` perché tanto quest'ultima è fuori del monitor all'estrema sinistra). L'istruzione `WAIT` è usata anche per terminare la COPPERLIST: infatti per indicare la fine della COP va messo un

```
1       dc.w    $FFFF,$FFFE     ; Fine Copperlist
```

Che per convenzione il Copper considera la fine, anche perché indica di attendere una linea che non esiste! (la copperlist poi riparte da capo). Si è sparsa la voce tempo fa che sarebbero necessarie due istruzioni di fine copperlist anziché una sola per alcuni vecchi modelli di Amiga, ma sembra sia una psicosi di massa, dato che nessuno ne ha mai messe due e ha sempre funzionato tutto.

Un ultima cosa: per fare la nostra copperlist che per ora è priva di disegni, ha solo sfumature, bisogna disabilitare i BITPLANE, ovvero i PIANI di BIT che sovrapponendosi danno luogo alle figure. per fare questo basta aggiungere la linea `DC.W $100,$200`, ossia mettiamo il valore `$200` nel `$dff100`, che è il registro di controllo dei bitplane.

Ora siamo in grado di fare completamente la copperlist che aspetta la metà del video e cambia il colore!

```
1   COPPERLIST:
2           dc.w    $100,$200       ; BPLCON0 Nessuna figura, solo lo sfondo
3           dc.w    $180,0          ; Color 0 NERO
4           dc.w    $7f07,$FFFE     ; WAIT - Aspetta la linea $7f (127)
5           dc.w    $180,$00F       ; Color 0 BLU
6           dc.w    $FFFF,$FFFE     ; FINE DELLA COPPERLIST
```

Considerando che per verificare il funzionamento delle vostre copperlist dovrete fare delle sfumature di colore, ecco una *tabella di riferimento per la scelta dei colori del copper*.

L'Amiga ha 32 registri colore per 32 colori diversi:

```
1           $dff180         ; color0 (sfondo)
2           $dff182         ; color1
3           $dff184         ; color2
4           $dff186         ; color3
5           ...
6           $dff1be         ; color31
```

In ognuno si questi 32 registri colore si può selezionare uno dei 4096 colori visualizzabili, "mischiando" i 3 colori fondamentali ROSSO, VERDE, BLU. Ognuno di questi 3 colori può avere una intensità da 0 a 15, ossia 16 toni. Infatti il massimo numero di combinazioni è 16*16*16=4096, ossia 16 ROSSI moltiplicato 16 VERDI moltiplicato 16 BLU. Il valore del colore si può mettere col processore o col COPPER:

```
1           move.w  #$000,$dff180   ; colore NERO in color0
2
3           dc.w    $180,$FFF       ; colore BIANCO in color0
```

In questo esempio abbiamo visto i due estremi: `$FFF`, ossia BIANCO, e `$000`, ossia NERO. Per scegliere il colore infatti occorre tener presente che la WORD del colore è composta così:

```
1           dc.w    $0RGB
2
3           dove il quarto zero è inutilizzato, mentre:
4
```

```
5       R       =       componente ROSSA (RED)
6       G       =       componente VERDE (GREEN)
7       B       =       componente BLU   (BLU)
```

Infatti i bit dal 15 al 12 non sono utilizzati, i bit dall'11 all'8 sono il ROSSO, quelli dal 7 al 4 sono il VERDE, quelli dal 3 allo 0 sono il BLU.

Ogni colore RGB come già detto può avere un valore da 0 a 15, ossia da 0 a $F in esadecimale, dunque è facile scegliere il colore:

```
$FFF    =       Bianco
$D00    =       Rosso mattone
$F00    =       Rosso
$F80    =       Rosso-Arancio
$F90    =       Arancione
$fb0    =       Giallo-oro
$fd0    =       Giallo-Cadmio
$FF0    =       Limone
$8e0    =       Verde chiaro
$0f0    =       Verde
$2c0    =       Verde scuro
$0b1    =       Verde albero
$0db    =       Acqua
$1fb    =       Acqua chiaro
$6fe    =       Blu cielo
$6ce    =       Blu chiaro
$00f    =       Blu
$61f    =       Blu brillante
$06d    =       Blu scuro
$c1f    =       Violetto
$fac    =       Rosa
$db9    =       Beige
$c80    =       Marrone
$a87    =       Marrone scuro
$999    =       Grigio medio
$000    =       nero
```

Ora il problema è solo come costringere il copper ad eseguire ordini dalla nostra COPPERLIST sviando la sua attenzione da quella del WorkBench; ma c'è anche un altro problema: se facciamo eseguire la nostra, come facciamo dopo essere usciti a fargli rileggere quella di sistema??? Risposta: Bisogna segnarsi su un foglietto dove era!!! Ovvero: lo segnamo in un apposita longword denominata `OLDCOP`, ovvero VECCHIA COPPERLIST, quella di sistema. Ma a chi lo dobbiamo chiedere dove è la copperlist di sistema? al sistema operativo ovviamente!! Per chiederglielo dovremo eseguire delle routines che sono nel CHIP del kickstart!!! Per fare questo bisogna sempre prendere come riferimento l'indirizzo che si trova nell'indirizzo *$4*, che viene scritto dal kickstart e contiene appunto l'indirizzo da cui si possono fare le distanze di indirizzamento prefissate, di cui parleremo in seguito. per raccogliere la long all'indirizzo *$4* basta fare un:

```
1       MOVE.L  $4,a6   ; In a6 ora abbiamo l'ExecBAse
```

O meglio

```
1       MOVE.L  4.w,a6  ; Infatti 4 è un numero piccolo e si può scrivere
2                       ; 4.w, il che risparmia spazio. (scrive l'istruzione
3                       ; con $0004 invece di scriverla con $00000004, in cui
4                       ; i primi zeri non servono. VIENE COMUNQUE SPOSTATA
5                       ; UNA LONGWORD! la long contenuta nei 4 bytes 4,5,6,7.
```

Messo l'indirizzo che era contenuto in *$4* in `a6`, possiamo eseguire le routines del kickstart facendo dei `JSR` con la distanza di indirizzamento giusta, infatti esistono delle distanze di indirizzamento precise che corrispondono a certe routines già pronte nel kickstart. Ora sappiamo

che se facciamo, ad esempio, un `JSR -$78(a6)` disabilitiamo il multitasking!!! Ovvero viene eseguito solo il nostro programma! facciamolo subito! Caricate `LEZIONE3a.s` in un buffer Fx ed eseguitelo.

Però la Exec non si occupa di tutto: il kickstart, lungo 256k se è la versione 1.2 o 1.3, oppure lungo 512k se è 2.0 o 3.0, è diviso in library, ossia delle "raccolte" di routine già pronte che possono essere chiamate, e siccome ogni kickstart è diverso proprio fisicamente, nel senso che ad esempio la routine della Exec che disabilita il sistema operativo nel kick 1.3 potrebbe essere a *$fc1000*, mentre nell'1.2 o nel 2.0 a diversi indirizzi ancora, i cari progettisti hanno avuto una delle loro clamorose idee: *"perché non mettiamo un indirizzo alla locazione $4 da cui si possa sempre eseguire la stessa routine facendo un `JSR` allo stesso offset (ovvero distanza di indirizzamento)?"* (P.S. `JSR` è come `BSR`, solo che può eseguire routines in qualsiasi parte della memoria, mentre il `bsr` le può eseguire se sono entro 32768 bytes in avanti o indietro).

Ed è questo quello che hanno fatto! Per eseguire ad esempio il Disable, che disabilita il sistema operativo, su tutti i kickstart basta fare:

```
1        move.l  4.w,a6          ; Indirizzo della Exec in a6
2        jsr     -$78(a6)        ; Disable - blocca multitask
3        bsr.w   mioprogramma
4        jsr     -$7e(a6)        ; Enable - riabilita multitask
```

In ogni kickstart la routine sarà ad un indirizzo diverso, ma facendo in questo modo siamo sempre sicuri di eseguire quella routine. Basta sapere tutte le distanze di indirizzamento delle varie routines del sistema operativo per eseguirle, ma a noi interessa soltanto di salvare l'indirizzo della copperlist di sistema, e per farlo dobbiamo rivolgerci ad una parte delle routines del kick che si chiama: `graphics.library`, ossia quella che si occupa della GRAFICA, sia chiaro solo a livello di sistema operativo, non a livello hardware. Per accedere alla libreria grafica va APERTA, ossia dobbiamo fare così:

```
1        move.l  4.w,a6          ; Execbase in a6
2        lea     GfxName,a1      ; Indirizzo del nome della lib da aprire in a1
3        jsr     -$198(a6)       ; OpenLibrary, routine della EXEC che apre
4                                ; le librerie, e da in uscita l'indirizzo
5                                ; di base di quella libreria da cui fare le
6                                ; distanze di indirizzamento (Offset)
7        move.l  d0,GfxBase      ; salvo l'indirizzo base GFX in GfxBase
8        ....
9
10   GfxName:
11       dc.b    "graphics.library",0,0  ; NOTA: per mettere in memoria
12                                       ; dei caratteri usare sempre il dc.b
13   GfxBase:                            ; e metterli tra "", oppure ''
14       dc.l    0
```

In questo caso abbiamo usato la routine della Exec *OpenLibrary* che richiede che sia messo in `A1` l'indirizzo del testo con il nome della libreria da aprire. Per esempio potevamo aprire altre librerie come `dos.library` per caricare dei file o simili, `intuition.library` per aprire finestre ecc. Una volta eseguita al ritorno da in `d0` l'indirizzo della libreria in questione, per intenderci un indirizzo come *GfxBase* da cui fare dei `JSR` con degli offset a proposito della grafica. Oltre ai `JSR`, sappiamo anche che, per esempio, l'indirizzo della COPPERLIST di sistema attuale è situata a $26 bytes dopo il *GfxBase*, quindi continueremo il nostro programma salvando quell'indirizzo in una label `OldCop`:

```
1        move.l  4.w,a6          ; Execbase in a6
2        lea     GfxName,a1      ; Indirizzo del nome della lib da aprire in a1
3        jsr     -$198(a6)       ; OpenLibrary, routine della EXEC che apre
4                                ; le librerie, e da in uscita l'indirizzo
5                                ; di base di quella libreria da cui fare le
6                                ; distanze di indirizzamento (Offset)
7        move.l  d0,GfxBase      ; salvo l'indirizzo base GFX in GfxBase
8        move.l  d0,a6
9        move.l  $26(a6),OldCop  ; salviamo l'indirizzo della copperlist
```

```
10          ....                    ; di sistema
11
12  GfxName:
13          dc.b    "graphics.library",0,0  ; NOTA: per mettere in memoria
14                                          ; dei caratteri usare sempre il dc.b
15  GfxBase:                                ; e metterli tra "", oppure ''
16          dc.l    0
17
18  OldCop:
19          dc.l    0
```

Ora possiamo puntare la nostra copperlist, possiamo mettere un `MouseWait` e dopo ristabilire la vecchia cop; per puntare intendo mettere l'indirizzo della nostra copperlist nel registro `COP1LC`, ossia *$dff080*, che è il puntatore alla copperlist nel senso che il copper esegue la copperlist il cui indirizzo si trova in *$dff080*: basterà dunque mettere l'indirizzo in *$dff080*, poi per far partire la copperlist basta scrivere nel registro *$dff088* (`COPJMP1`) qualsiasi cosa, basta che ci si scriva o che si legga che fa partire la copperlist, è un registro detto **strobe**, come fosse un bottone che basta toccarlo (**non usate però** `CLR.W $dff088`**, da dei problemi**).

Verrà così eseguita ripetutamente ogni fotogramma la nostra copperlist fino a che non ne sarà messa un'altra nel *$dff080* (`COP1LC`). Un problema è che il `$dff080` è a sola scrittura, infatti se provare a fare un *<=c 080>* noterete il W di WRITE. Per poter rimettere a posto la copperlist di sistema, quella che visualizza l'asmone o il workbench, non potendone leggere l'indirizzo dal `$dff080`, dovremo chiedere al sistema operativo quale ci ha messo, e questo si può fare con delle routine del kickstart: una volta ottenuto l'indirizzo di quella copperlist lo salveremo in una LONGWORD del nostro programma, poi punteremo la nostra copperlist, e all'uscita del programma rimetteremo a posto quella vecchia.

```
1           move.l  4.w,a6              ; Execbase in a6
2           jsr     -$78(a6)            ; Disable - ferma il multitasking
3           lea     GfxName,a1          ; Indirizzo del nome della lib da aprire in a1
4           jsr     -$198(a6)           ; OpenLibrary, routine della EXEC che apre
5                                       ; le librerie, e da in uscita l'indirizzo
6                                       ; di base di quella libreria da cui fare le
7                                       ; distanze di indirizzamento (Offset)
8           move.l  d0,GfxBase          ; salvo l'indirizzo base GFX in GfxBase
9           move.l  d0,a6
10          move.l  $26(a6),OldCop      ; salviamo l'indirizzo della copperlist
11                                      ; di sistema
12          move.l  #COPPERLIST,$dff080     ; COP1LC - Puntiamo la nostra COP
13          move.w  d0,$dff088              ; COPJMP1 - Facciamo partire la COP
14  mouse:
15          btst    #6,$bfe001
16          bne.s   mouse
17
18          move.l  OldCop(PC),$dff080      ; COP1LC - Puntiamo la cop di sistema
19          move.w  d0,$dff088              ; COPJMP1 - facciamo partire la cop
20
21          move.l  4.w,a6
22          jsr     -$7e(a6)            ; Enable - riabilita il Multitasking
23          move.l  gfxbase(PC),a1      ; Base della libreria da chiudere
24                                      ; (vanno aperte e chiuse le librerie!!!)
25          jsr     -$19e(a6)           ; Closelibrary - chiudo la graphics lib
26          rts
27
28  GfxName:
29          dc.b    "graphics.library",0,0  ; NOTA: per mettere in memoria
30                                          ; dei caratteri usare sempre il dc.b
31  GfxBase:                                ; e metterli tra "", oppure ''
32          dc.l    0
33
34  OldCop:
35          dc.l    0
36
37  COPPERLIST:
38          dc.w    $100,$200           ; BPLCON0 - Nessuna figura, solo lo sfondo
39          dc.w    $180,0              ; Color 0 NERO
40          dc.w    $7f07,$FFFE         ; WAIT - Aspetta la linea $7f (127)
```

```
41          dc.w    $180,$00F       ; Color 0 BLU
42          dc.w    $FFFF,$FFFE     ; FINE DELLA COPPERLIST
```

Troverete questo esempio con suggerimenti e modifiche in `Lezione3b.s` Caricatevelo nel buffer *<F2>* o un altro qualsiasi ed ammirate il primo programma del corso che "BATTE NEL METALLO" dei CHIP dell'Amiga.

Avete fatto i vostri esperimenti sulla copperlist? Bene, ora vediamo di fare qualche effetto in movimento. Per cominciare però devo informarvi che per fare un qualsiasi movimento bisogna sincronizzare le routines con il pennello elettronico che ridisegna lo schermo. Per chi non lo sapesse infatti lo schermo viene ridisegnato 50 volte al secondo, e i movimenti che ci appaiono fluidi, ad esempio quelli dei videogames meglio programmati, sono spostamenti che coincidono con il cinquantesimo di secondo. Abbiamo già usato il registro `$dff006`, che come abbiamo visto cambia di valore continuamente, proprio perché c'è la posizione del pennello elettronico, il quale parte da zero, ossia dalla parte più alta dello schermo, e arriva in fondo 50 volte al secondo. Se facciamo una routine che fa dei movimenti sul video senza temporizzarla, andrà alla velocità effettiva del processore, dunque troppo veloce per vedere qualcosa. Per attendere una certa linea video basta leggere il primo byte del `$dff006`, in cui troviamo la linea raggiunta, ossia la posizione verticale (uguale al `WAIT` del COPPER):

```
1   WaitLinea:
2           CMPI.B  #$f0,$dff006    ; VHPOSR - Siamo alla linea $f0? (240)
3           bne.s   WaitLinea       ; se no, ricontrolla
4           ...
```

questo ciclo aspetta la linea 240, dopodiché l'esecuzione continua con le istruzioni seguenti, come la routine del Mouse che aspetta la pressione del tasto, dopodiché continua l'esecuzione. Inseriamo anche il `WaitMouse`:

```
1   mouse:
2           cmpi.b  #$f0,$dff006    ; VHPOSR - Siamo alla linea 240?
3           bne.s   mouse           ; Se non ancora, non andare avanti
4
5           bsr.s   RoutineTemporizzata     ; Questa routine viene eseguita 1
6                                           ; volta sola per ogni fotogramma
7
8           bsr.s   MuoviCopper     ; Il primo movimento sullo schermo!!!!!
9           btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
10          bne.s   mouse           ; se no, torna a mouse:
11          rts
```

A questo punto abbiamo una routine che esegue una routine 1 sola volta per ogni FRAME video, ossia per ogni fotogramma, ossia 1 volta ogni cinquantesimo di secondo, e più esattamente viene eseguita non appena siamo arrivati alla linea 240, dopodiché, una volta eseguita, non sarà eseguita nuovamente fino a che non saremo nuovamente alla linea 240, il fotogramma successivo.

NOTA: L'immagine viene disegnata con la tecnica **raster** tramite un pennello elettronico, che parte a disegnare dalla prima linea in alto a sinistra, prosegue verso destra fino alla fine della linea, poi riparte dall'estrema sinistra della linea 2, per andare verso destra ecc, analogamente al percorso che facciamo noi per leggere: ogni linea da sinistra verso destra, partendo dalla prima in altro fini all'ultima in basso, dopodiché il pennello elettronico riparte dalla prima linea, primo punto a sinistra, come se noi avendo finito di leggere una pagina di libro ricominciassimo a leggerla anziché leggere la pagina seguente. D'altronde il monitor è uno solo e deve scrivere su quello solamente, il pennello non scrive sul muro.

Caricatevi l'esempio `LEZIONE3c.s` in un altro buffer di testo e provatelo. Questo esempio fa muovere in basso una `WAIT` e quindi il colore seguente quando premete il tasto destro del mouse. Tasto sinistro per uscire.

Avete compreso `Lezione3c.s`? Allora complichiamo leggermente le cose! Caricate la `Lezione3c2.s` in un buffer e studiatelo, ho aggiunto un controllo della linea raggiunta per fermare lo scroll.

Tutto chiaro in `Lezione3c2.s`?? Bene, continuiamo con la pratica caricando la `Lezione3c3.s`, in cui viene spostata una barretta sfumata fatta con 10 wait anziché una sola linea `WAIT`. Sempre più difficile!!!

Siete sempre vivi dopo la `Lezione3c3.s`? Massacratevi il cervello con la lezione seguente, la `Lezione3c4.s`, in cui passiamo da 10 label BARRA ad una sola label eseguendo delle distanze di indirizzamento.

Beh, non era poi così difficile. Il difficile viene ora con `Lezione3d.s`, in cui la barra va su e giù, e cambieremo anche la velocità della barra.

Avete capito `Lezione3d.s`? Si? Non ci credo! Vi sembra di aver capito, non può essere... io lo rivedrei un attimo prima di proseguire... lo avere rivisto? Beh... allora caricatevi una variazione sul tema, `Lezione3d2.s`

Ora siete pronti per affrontare la `Lezione3e.s`, in cui è spiegato come fare una RASTERBAR ossia un effetto di scorrimento ciclico dei colori.

Un altro caso particolare: Come si fa a raggiungere la zona PAL (dopo $FF) con i wait del copper in `Lezione3f.s`.

Per completare la `lezione3.txt`, caricatevi la `Lezione3g.s`, e `Lezione3h.s`, concernenti uno scorrimento a destra e sinistra anziché in basso ed in alto, dopodiché sarete pronti per la `LEZIONE4.TXT`, in cui sarà trattata la gestione delle immagini colorate e dei possibili effetti su di esse!

NOTA: Gli `Esempi4x.s` della `LEZIONE4.TXT` si trovano nella directory `SORGENTI2`, dunque dovete fare un *<V DF0:SORGENTI2>* per rendere possibile il caricamento delle immagini da quella directory. Dopodiché caricate la `LEZIONE4.TXT` in questo o in un altro buffer di testo. (con *<r>*)

Complimenti per essere arrivati qua! Il grosso è fatto! Ora andando avanti capirete con facilità, essendo entrati nella logica della programmazione ASM!.

CAPITOLO 4

LEZIONE 4 - IMMAGINI SULLO SCHERMO

In questa lezione impareremo a visualizzare delle figure in varie risoluzioni tramite la copperlist. Fino ad ora abbiamo potuto cambiare solo il `color0`, ossia il *$dff180*, con cui abbiamo fatto delle sfumature, ma chiaramente le figure non si fanno a forza di `WAIT`!!! Per visualizzare una normale figura IFF, creata col Deluxe Paint, digitalizzata, scannerizzata o renderizzata con un programma di ray tracing quali Imagine o Real 3d non serve alcun `WAIT`!!!! Basta dire al copper che risoluzione grafica ha la figura (numero di colori, risoluzione hires o lowres, interlacciata o meno) tramite il registro `BPLCON0`, ossia il *$dff100*, che per ora abbiamo sempre mantenuto col valore `$200`, che indica: *solo il colore di sfondo senza immagini in "sovraimpressione"*. è per questo che se in una copperlist del genere cambiamo ad esempio il `color1`, ossia *$dff182*, non accade nulla: perché non è abilitato nessun *bitplane*, ossia "piano di bit", ma solo lo "sfondo", il cui colore può essere cambiato col *$dff180*. Dopo aver definito il numero di colori e la risoluzione grafica (ad esempio 320x200 pixel, dove per PIXEL si intende ognuno dei piccoli punti che compongono la figura), avendo indicato dove si trova la figura da visualizzare mettendo il suo indirizzo negli appositi *puntatori* (registri come il `$dff080` (`COP1LC`) dove invece viene messo l'indirizzo dei bitplane), bisogna definire i colori della figura, ossia la *palette* (cioè la "tavolozza" dei colori definita dal programma di disegno (es. DeLuxe Paint) per la figura in questione), oppure la figura apparirà con i colori sbagliati. In pratica dobbiamo mettere in copperlist i registri colore necessari, se la figura è a 4 colori dobbiamo definire i 4 colori:

```
1       dc.w    $180,$xxx       ; color 0
2       dc.w    $182,$xxx       ; color 1
3       dc.w    $184,$xxx       ; color 2
4       dc.w    $186,$xxx       ; color 3
```

Questo pezzo di copperlist comunque lo salva direttamente il KEFCON

Ci sono anche altri *registri* che regolano la dimensione della figura per farla di dimensioni "speciali", come l'*overscan* che la rende più grande, o si può fare una "finestrella" che occupi solo una parte dello schermo. Altri registri speciali sono i *moduli*, usati spesso per gli effetti di allungamento delle figure. Nei primi esempi comunque manterremo i registri speciali azzerati o comunque con i valori normali per visualizzare una figura. Innanzitutto deve essere chiaro che c'è differenza tra un file IFF, ossia una figura nel formato standard caricabile dal DeLuxe Paint, e la figura REALE (detta RAW o BITMAP) che sta in memoria e viene visualizzata dal copper. Nel

disco è presente un programmino capace appunto di convertire figure in formato IFF nel formato RAW indispensabile per visualizzare le figure direttamente con il COPPER. Le figure in realtà sono composte di molti 0 ed 1, come tutti i dati BINARI in memoria. Abbiamo già visto che ogni dato in memoria è composto di BIT, ossia di zeri e di 1, che sono gli unici 2 stati possibili della memoria (essendo possibile solo *passaggio di corrente* e *assenza di corrente*); per comodità usiamo il sistema decimale e esadecimale, ma la realtà è sempre quella dei BIT. Allora come è possibile visualizzare una figura a 32 colori quando i bit possono essere solo 0 ed 1??? Se mettessimo in memoria una specie di foglio di carta a quadretti con dei quadretti anneriti (ossia 1) ed altri bianchi (ossia zero) potremmo solamente lavorare con 2 colori, il nero e il bianco, come i vecchi computer con i monitor a fosfori verdi che potevano visualizzare solo il colore di sfondo (i bit a zero) e certi disegni o parole fatte di bit accesi (ad 1). Con il COPPER si può anche operare in questo modo, a 2 colori, basta *accendere un solo bitplane*. In questo caso dovremo mettere in memoria la figura RAW, composta similmente al foglio di carta millimetrata descritto prima, con dei punti "accesi" e dei punti "spenti". Fino qui tutto dovrebbe essere chiaro: è come fare la battaglia navale! Una nave sarebbe per esempio fatta di un certo numero di PIXEL (punti), allo stesso modo si può fare qualsiasi cosa:

```
 UN UOMO:                                UN AEREO (ho risparmiato gli zeri!)
                                                  11
000011100000          000001100000              1111
000001000000          000010010000              1111
000111111000          000010010000        11111111111111111111
000101101000          000111111000       1111111111111111111111111
000101101000          000100001000              1111
000011110000          000100001000              1111
000010010000                                   111111
000010010000             UNA "A"              11111111
000010010000
000110011000
```

Quando la figura è grande conviene chiaramente farsela col programma di disegno o scannerizzarla, poi convertirla in RAW (0000110101...) col programma in questo disco (KEFCON). Per definire il colore di sfondo (gli zeri) basterà mettere un `dc.w $180,$000` (nero), per definire il colore degli 1 basterà mettere un `dc.w $182,$0f0` (verde). Per le figure a più colori il trucco è questo: i vari *bitplanes*, ossia *piani di bit* (0001010 ecc.) vengono "SOVRAPPOSTI" in una specie di *trasparenza*, per cui dove vengono sovrapposti due "1" appare un certo colore, dove sono sovrapposti tre "1" appare un altro colore eccetera. Tutto questo non va calcolato!!! Basta caricarsi la figura con l'Iffconverter e salvarla in RAW, poi mettere il numero dei colori e la risoluzione nel `$dff100` (`bplcon0`), dire al copper dove abbiamo messo la figura RAW, mettere i colori giusti (che salva direttamente l'iffconverter separatamente), e la figura appare senza problemi. L'importante è aver chiaro il procedimento, in pratica poi per convertire l'immagine IFF in RAW e modificare il sorgente ci vogliono due minuti.

Innanzitutto chiariamo cosa fa l'iffconverter (nel nostro caso usiamo quello chiamato KEFCON, che potete caricare spostando la finestra dell'asmone e scrivendo nel MENU del DOS il suo nome; esistono iffconverters più recenti e con più opzioni, alcune delle quali spesso inutili, ma per ragioni di spazio e di compatibilità con kickstart 1.3 ho deciso di mettere questo nel corso, poi c'è anche il fatto che è programmato usando i registri hardware anziché il sistema operativo, dunque è in linea col corso. Se volete usare altri iffconverters fate pure, ma prima imparate ad usare questo, che è stato usato per fare giochi e demo gloriosi). Una immagine nella realtà abbiamo visto che è un insieme di piani di bit, più piani per più colori, un piano solo per 2 colori. Abbiamo anche visto che per essere visualizzata servono i colori giusti (la palette) e la risoluzione giusta nel `$dff100` (`bplcon0`); i programmatori dell'Amiga decisero di creare un formato

particolare per memorizzare le figure e trasferirle da un programma all'altro: questo formato per l'Amiga è l'IFF ILBM, ed è in pratica composto da i bitplanes compattati con una certa routine per prendere meno spazio, con attaccata la palette e la risoluzione; un programma quando carica un immagine IFF scompatta i PIANI compattati, mette la palette a posto nei registri dei colori (`$dff180`, `$dff182`, `$dff184` eccetera) e la risoluzione nel `$dff100`, il `BPLCON0` (Sommariamente). Allo stesso modo quando ha una immagine in memoria, per salvarla in IFF deve comprimere i BITPLANES nel formato IFF, attaccarci la palette e il resto. L'Iffconverter fa queste operazioni: può caricare un RAW, e salvarlo in IFF, a patto che gli si dia la PALETTE e la RISOLUZIONE giusta, oppure può caricare un immagine IFF e salvarla in RAW, e può salvare la PALETTE già nel formato `dc.b $180,xxx,$182,xxx`, ossia la PALETTE già da inserire nella copperlist. In altri computer si utilizzano diversi formati per le figure, ad esempio il GIF, il PCX e il TIFF sono usati dai PC MSDOS; oltre ad essere diversamente COMPRESSE le figure RAW+PALETTE, in questi computer è diverso anche il sistema di visualizzazione, infatti hanno il sistema **chunky** anziché bitplane, un sistema utile quando si debbono gestire 256 colori, ma meno capace di quello Amiga per quanto riguarda gli *scroll* (scorrimenti) e senza possibilità di cambiare la PALETTE come fa il COPPER con i WAIT. Le possibili risoluzioni grafiche dell'Amiga normale (non AGA) sono queste:

- 320x256 PIXEL detta LOW RES (bassa risoluzione)
- 640x256 PIXEL detta HIGH RES (alta risoluzione)

L'immagine può essere anche più lunga (312 linee in overscan) oppure lunga il doppio (tramite l'interlace che però causa uno sfarfallamento). Anche la larghezza può essere leggermente aumentata con l'*overscan*.

Le immagini in LOW RES (larghe 320 pixel) possono avere fino ad un massimo di 32 colori, ci sono poi 2 modi speciali detti *EHB* (Extra Half Bright) e *HAM* (Hold And Modify) che visualizzano rispettivamente 64 e 4096 colori, ma hanno delle limitazioni particolari che vedremo. Le immagini in *high res* (alta risoluzione) possono avere un massimo di 16 colori e non dispongono di modi speciali. I giochi sono quasi tutti in *low res*, per sfruttare il maggior numero di colori disponibili, per il risparmio di memoria (che purtroppo FINISCE!), e per la maggiore velocità raggiungibile (infatti l'HIGH RES rallenta le operazioni maggiormente del LOW RES, inoltre bisognerebbe spostare pezzi più grandi di memoria essendo l'immagine larga il doppio!).

Analizziamo la tecnica di visualizzazione dei colori: abbiamo detto che il massimo dei colori in lowres è 32 (senza contare i modi speciali); è possibile scegliere una risoluzione video con 2,4,8,16 o 32 colori. Questo perchè sono determinati dalla sovrapposizione di BIT, dunque ogni bitplane che "sovrapponiamo" aggiungiamo 2 bit ad ogni "pixel" che diventa *più profondo* di 2 bit: ora con 2 bit si può ottenere soltanto 0 ed 1, ossia 2 colori, allora la risoluzione 320x200 a 2 colori avrà un solo BITPLANE, come abbiamo già detto. Aggiungendo un altro bitplane i colori possibili diventano 4, infatti si possono verificare 4 situazioni di sovrapposizione per ogni PIXEL: `00, 01, 10, 11` ossia "entrambi i bit a zero", "primo bitplane a zero e secondo ad 1", "primo bitplane ad 1 e secondo a zero", "entrambi i bitplane ad 1". Aggiungendo un altro bitplane si possono verificare 8 situazioni diverse (corrispondenti a 8 colori diversi):

```
000,001,010,011,100,101,110,111   (3 bitbplanes=3 bit per PIXEL=8 possibilità)
```

Aggiungendo un bitplane, il quarto, arriviamo ad 4 bit possibili per PIXEL, ossia 16 diverse possibilità per 16 colori:

```
0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111
```

Lo stesso vale per il quinto bitplane, che porta a 5 il numero di bit per pixel, ossia 32 colori possibili. Dunque ogni bitplane eleva alla seconda il numero dei colori:

- 0 bitplane = solo il colore di sfondo COLOR00 ($dff180), 1 COLORE
- 1 bitplane = 2 COLORI
- 2 bitplane = 4 COLORI (2*2, ossia 2 alla seconda)
- 3 bitplane = 8 COLORI (2*2*2, ossia 2 alla terza)
- 4 bitplane = 16 COLORI (2*2*2*2, ossia 2 alla quarta)
- 5 bitplane = 32 COLORI (2*2*2*2*2, ossia 2 alla quinta)

L'Amiga ha 32 registri per i 32 colori possibili in LOWRES, che partono dal `COLOR0` per arrivare fino al `COLOR31` (la numerazione parte contando lo zero come per i bit). Il `color0` è il `$dff180`, seguito dagli altri:

- `$dff182` = `COLOR1`
- `$dff184` = `COLOR2`
- `$dff186` = `COLOR3`
- `$dff188` = `COLOR4`
- `$dff18a` = `COLOR5`
- eccetera...

per esempio se un pixel di una immagine lowres a 16 colori è del colore dello "SFONDO", ossia del `COLOR0`, modificabile agendo sul `$dff180`, vuol dire che tutti e 4 i bitplanes sono a zero: `0000`, mentre un pixel che ha il colore 32, detto `COLOR31`, sarà il risultato di questa combinazione binaria: `1111`. Gli altri colori sono il frutto delle altre combinazioni. L'amiga 1200 dispone di 8 bitplanes al massimo con il suo *chipset AGA*, infatti può generare 256 colori (2 all'ottava=256), infatti nei programmi grafici AGA si possono scegliere anche risoluzioni con 64 colori (6 planes), 128 colori (7 planes). Una schermata video è detta anche **playfield**.

Vediamo di calcolare quanta memoria occupa una figura a 2 colori in 320*256: ogni linea ha 320 pixel, essendo un byte composto di otto bit, in una linea ci sono 40 byte (infatti 8*40=320). Allora basterà moltiplicare 40, ossia il numero di byte per linea, per il numero delle linee, cioè 256: 40*256=10240. Dunque un bitplane in lowres occupa 10240 bytes. Allora possiamo calcolare anche una figura con 4 colori, ossia 2 bitplanes: 40*256*2=20480. Dunque per una figura in LOWRES standard basta moltiplicare 40*256*bitplanes. Stabilito che in LOWRES ci sono 320 bit per linea, ossia 40 bytes, in HIRES essendo largo il doppio ci saranno 80 bytes per linea: 80*256*bitplanes. In definitiva, la formula generale per calcolare la grandezza è:

```
Byte per linea * linee del playfield * numero bitplane
```

Analizziamo ora il `BPLCON0`, il registro dove va indicata la risoluzione e il numero dei colori: (Potete leggere un riassunto scrivendo *<=C 100>*)

```
        $dff100 - BPLCON0

 Bit Plane Control Register 0   (1 word, ossia 16 bit, dallo 0 al 15)

 NUMERO DEL BIT                 (nota: bit ad 1 = ON, bit a 0 = OFF)
```

```
15      -       HIRES   Modo hires (1=640x256 , 0=320x256)
14      -       BPU2    \
13      -       BPU1     ) 3 bit per scegliere il numero di bitplanes
12      -       BPU0    /
11      -       HOMOD   Hold And Modify mode (HAM 4096 colori)
10      -       DBLPF   Double playfield
09      -       COLOR   video composito (per il GENLOCK)
08      -       GAUD    Genlock audio
07      -       X
06      -       X
05      -       X
04      -       X
03      -       LPEN    Lightpen (Penna ottica)
02      -       LACE    Interlacciato (320x512 o 640x512)
01      -       ERSY    External resync (Per il GENLOCK)
00      -       X
```

Questo registro è *bitmapped*, ossia ogni suo bit ha una funzione:

- Il bit 15 abilita il modo hires: questo modo grafico visualizza 640 pixel per linea orizzontale invece di 320. Ricordatevi di mettere `DDFSTART/STOP` a `$003c` e `$00d4` anziché `$0038` e `$00d0`, altrimenti non verranno visualizzate le prime linee del bordo sinistro!

- I bit 14-12 servono a stabilire il **numero** di plane da accendere, **non** quali plane; infatti i bit sono 3 ed i plane possibili sono 6. Qui bisogna scrivere **quanti** plane da accendere, proprio come un numero, **non** selezionando **quali**. Es.: '3', '0', '6'. In 3 bit, infatti, sono esprimibili 8 numeri, dallo 0 al 7. Ripeto: **lavorate con un numero vero e proprio in binario a 3 bit, non con singoli bit da accendere o spegnere, a differenza degli altri bit!** N.B.: Scrivendo '0' (=%000) si spengono tutti i plane, scrivendo %101 se ne accendono 5; con 6 plane si attiva l'*Half-Bright* mode a 64 colori.

- Il bit 11 serve ad azionare il modo *[HAM]* (vanno accesi 6 plane) L'HAM consente agli amiga normali di visualizzare 4096 colori, l'HAM8 permette agli Amiga 1200/4000 di visualizzarne 262144.

- Il bit 10 consente di attivare il modo *Dual PlayField*, uno speciale modo a 2, 4 o 6 plane che permette di creare due schermi da 1, 2 o 3 plane ciascuna, chiamati PlayField1 e PlayField2, accavallati l'uno sull'altro in trasparenza renedendo trasparente il colore 0 del PlayField sovrastante. È possibile creare, quindi, un effetto parallattico simile a quello presente in molti giochi. Per esempio, si potrebbe utilizzare un PlayField a 3 plane (8 colori) per l'area di gioco ed un altro PlayField di sfondo, che magari scrolla (scorre) più lentamente per dare un maggiore senso di profondità, raffigurante pianure e montagne. Appena settato il bit i plane dispari (1, 3, 5) formano il *Playfield1*, mentre quelli pari (2, 4, 6) il secondo: l'hardware, appena attivato il bit `DPF`, raggruppa così i plane per poterli rendere indipendenti, dato che, come vedremo, esistono registri di scroll ed altri che distinguono parametri per plane pari e dispari, usati anche per controllare indipendentemente due interi playfield in dual mode! N.B.: Il modo DualPlayField consente di sovrapporre solo 2 schermi e, comunque, di analoga risoluzione grafica (es.: Hires+Hires, Lowres+Lowres, ecc...).

- Il bit 9 serve ad attivare anche l'uscita videocomposita degli Amiga posta accanto a quella RGB del monitor. Personalmente, la attivo sempre per consentire di vedere qualcosa sul monitor durante i miei prodotti anche a chi non possiede un monitor RGB standard. **Settatelo sempre a 1**.

- Il bit 8 attiva l'audio di un eventuale genlock collegato all'Amiga: non serve praticamente a niente, sorvoliamo.
- Il bit 7 è usato solo dai chipset evoluti dell'A1200, su amiga normale non ha funzioni. Ricordatevi comunque di lasciare sempre a zero questi bit inutilizzati, altrimenti su a1200 rischiate di compromettere il funzionamento del vostro demo/gioco/programma.
- Il bit 6 non ha funzioni su amiga normale, lasciatelo a zero.
- Il bit 5 lasciatelo a zero
- Il bit 4 lasciatelo a zero
- Il bit 3 serve a far ricevere le coordinate della penna ottica nei registri `VHPOS` (`$dff006`) e `VPOS` (`$dff004`) del pennello elettronico. La penna ottica su Amiga non è utilizzata quasi mai, non interessa questa opzione.
- Il bit 2 imposta il modo InterLace, che consente la visualizzazione di una videata con doppia risoluzione verticale, ma interlacciata. (512 linee) Vedremo in seguito come funziona questa modalità
- Il bit 1 serve a sincronizzare il movimento del pennello con la frequenza di qualche apparecchio esterno all'Amiga, dunque lasciatelo sempre a zero.
- Il bit 0 lasciatelo a zero

Detto questo facciamo degli esempi sull'uso del `$100` (`BPLCON0`) in copperlist:

```
1           ; 5432109876543210
2  dc.w $100,%0100001000000000    ; ---> 4 plane in Lowres (320x256)
3  dc.w $100,%1011001000000100    ; ---> 3 plane in Hires+Interlace (640x512)
4  dc.w $100,%0110001000000100    ; ---> 6 plane in HALF-BRIGHT Lowres+Lace
5  dc.w $100,%0110101000000000    ; ---> 6 plane in HAM lowres (4096 colori)
6  dc.w $100,%0110011000000000    ; ---> DualPlayField 3+3 plane in Lowres
7  dc.w $100,%1100011000000100    ; ---> DualPlayField 2+2 in Hires+interlace
```

Nella lezione 3 abbiamo usato il `BPLCON0` nella copperlist dandogli valore `$200`:

```
1         dc.w    $100,$200
```

Infatti abbiamo settato solo il bit 9, che serve per attivare il genlock:

```
1                 ; 5432109876543210
2         dc.w    $100,%0000001000000000
```

Il genlock è un apparecchio che serve per mettere in sovraimpressione titoli o grafica fatta con l'amiga su video televisivi, quindi chi non ha questo accessorio non vede cambiamenti tra una copperlist con il bit 9 attivato ed una con il bit disattivato, ma conviene tenerlo sempre ad 1, per chi volesse usare il genlock con le nostre copperlist, e perché il vecchio Amiga 1000 ha una uscita videocomposita per il monitor a colori. Dunque avremmo avuto lo stesso risultato in RGB anche con un `dc.w $100,0`. Come vedete, i bitplane sono ZERO, dunque c'è solo il colore di sfondo senza figure in sovraimpressione. Per "accendere" dei bitplane basta mettere il numero di bitplanes che vogliamo, in binario, nei bit 12, 13, 14 del registro.

Per esempio, per fare uno schermo con 1 bitplane (2 colori): (320x256!)

```
1                 ; 5432109876543210
2         dc.w    $100,%0001001000000000  ; BPLCON0 - bit 12 acceso!! (1 = %001)
```

*

Per uno schermo a 2 bitplanes: (4 colori)

```
                    ; 5432109876543210
        dc.w    $100,%0010001000000000  ; BPLCON0 - bit 13 acceso!! (2 = %010)
```

*

Per uno schermo a 3 bitplanes: (8 colori)

```
                    ; 5432109876543210
        dc.w    $100,%0011001000000000  ; bits 13 e 12 accesi!! (3 = %011)
```

*

Per uno schermo a 4 bitplanes: (16 colori)

```
                    ; 5432109876543210
        dc.w    $100,%0100001000000000  ; BPLCON0 - bit 14 acceso!! (4 = %100)
```

*

Per uno schermo a 5 bitplanes: (32 colori)

```
                    ; 5432109876543210
        dc.w    $100,%0101001000000000  ; bits 14,12 accesi!! (5 = %101)
```

Per uno schermo a 6 bitplanes: (per modi speciali EHB e HAM 4096 colori)

```
                    ; 5432109876543210
        dc.w    $100,%0110001000000000  ; bits 14,13 accesi!! (6 = %110)
```

(In questa modalità se non si mette ad 1 il bit dell'HAM (11) lo schermo è in Extra Half Bright, se il bit 11 invece è settato lo schermo è HAM.

Dunque basta mettere il numero di bitplanes richiesto nei 3 bit 12, 13, 14 del registro. Se si desidera uno schermo in hires, largo 640 pixel anzichè 320, basterà mettere ad 1 il bit 15, il primo a sinistra, **ricordandosi** che il massimo numero di bitplanes in HIRES è 4 (16 colori), e che bisogna cambiare il `DFFSTART` (`$dff092`) e il `DFFSTOP` (`$dff094`) rispetto al LOWRES:

```
        dc.w    $92,$003c       ; DdfStart HIRES normale
        dc.w    $94,$00d4       ; DdfStop HIRES normale
```

Lo stesso vale per l'interlacciato (lunghezza 512 linee anziché 256), basta mettere ad 1 il bit 2.

Una volta impostato correttamente il `BPLCON0`, bisogna dire dove sono i bitplane che abbiamo "attivato". Per fare ciò basta mettere i loro indirizzi negli appositi registri, che sono:

```
$dff0e0 = BPL0PT (puntatore bitplane 1)
$dff0e4 = BPL1PT (puntatore bitplane 2)
$dff0e8 = BPL2PT (puntatore bitplane 3)
$dff0ec = BPL3PT (puntatore bitplane 4)
$dff0f0 = BPL4PT (puntatore bitplane 5)
$dff0f4 = BPL5PT (puntatore bitplane 6)
```

Anche qua si parte dallo zero, dunque si arriva al 5 per definire il sesto. Certe volte invece si può trovare indicato il `$dff0e0` con `BPL1PT`, e di conseguenza si arriva al 6 per definire il sesto. L'help dell'Asmone parte da `BPL0PT`, potete verificare digitando *<=c 0e0>*. Per visualizzare una figura dunque basta puntare una copperlist con il `BPLCON0` giusto e i colori giusti, poi bisogna *puntare* anche i bitplanes, ad esempio così:

```
        MOVE.L  #BITPLANE0,$dff0e0      ; indirizzo di BITPLANE0 in BPL0PT
        MOVE.L  #BITPLANE1,$dff0e4      ; BPL1PT
        MOVE.L  #BITPLANE2,$dff0e8      ; BPL2PT
        ...
```

E la figura apparirà magicamente. I bitplanes comunque sono puntati nella copperlist direttamente, in quanto i puntatori devono essere riscritti ogni fotogramma.

Non bisogna mai dimenticarsi di mettere nella copperlist i registri "speciali", che per ora useremo azzerati o comunque col valore normale, altrimenti rimangono con valore della copperlist

del workbench e la visualizzazione può essere disturbata se questi registri non erano azzerati (per esempio il workbench del kickstart 1.3 ha i MODULI azzerati, mentre il kickstart 2.0 li ha con valori diversi: i giochi e le demo grafiche che vanno sul kick 1.3 e invece visualizzano male le figure sul kick 2.0 spesso sono state fatte con copperlist dove mancavano i registri dei moduli, il `$dff108` e `$dff10a`, che su kick1.3 sono a zero, dunque funzionava al collaudo, ma su kick 2.0 sballano la visualizzazione. Per evitare questi problemi dunque in ogni copperlist è sempre bene definirsi tutti i registri, anche quelli che non usiamo; i registri in questione sono:

```
$dff08e - DIWSTRT, inizio finestra video - normalmente a $2c81
$dff090 - DIWSTOP, fine finestra video - normalmente a $2cc1
$dff092 - DDFSTRT, data fetch start - normalmente a $0038
$dff094 - DDFSTOP, data fetch stop - normalmente a $00d0
$dff102 - BPLCON1, Bitplane control 1 - normalmente a $0000
$dff104 - BPLCON2, Bitplane control 2 - normalmente a $0000
$dff108 - BPL1MOD, modulo bitplanes pari - normalmente a $0000
$dff10A - BPL2MOD, modulo bitplanes dispari - normalmente a $0000
```

Parleremo di questi registri quando li useremo per creare effetti speciali, per ora basta ricordarsi di mettere sempre all'inizio della copperlist questi registri con i valori standard:

```
1  COPPERLIST:
2          dc.w    $8e,$2c81       ; DiwStrt
3          dc.w    $90,$2cc1       ; DiwStop
4          dc.w    $92,$0038       ; DdfStart * NOTA: per HIRES 640x256 $003c
5          dc.w    $94,$00d0       ; DdfStop  * NOTA: per HIRES 640x256 $00d4
6          dc.w    $102,0          ; BplCon1
7          dc.w    $104,0          ; BplCon2
8          dc.w    $108,0          ; Bpl1Mod
9          dc.w    $10a,0          ; Bpl2Mod
10
11         dc.w    $100,xxxx       ; Bplcon0 - Definiamo i colori e la risuluzione
12
13 ;       Qua possiamo mettere i colori della figura; questo pezzo di
14 ;       copperlist lo genera direttamente l'iffconverter KEFCON, la
15 ;       salva a parte con un nome a piacere, successivamente si può
16 ;       includere qua con l'opzione TAGLIA&INCOLLA dell'editor caricandolo
17 ;       in un altro buffer, ad esempio.
18
19         dc.w $0180,$0010,$0182,$0111,$0184,$0022,$0186,$0222
20         dc.w $0188,$0333,$018a,$0043,$018c,$0333,$018e,$0154
21         dc.w $0190,$0444,$0192,$0455,$0194,$0165,$0196,$0655
22         dc.w $0198,$0376,$019a,$0666,$019c,$0387,$019e,$0766
23         dc.w $01a0,$0777,$01a2,$0598,$01a4,$0498,$01a6,$0877
24         dc.w $01a8,$0888,$01aa,$05a9,$01ac,$0988,$01ae,$0999
25         dc.w $01b0,$06ba,$01b2,$0a9a,$01b4,$0baa,$01b6,$07cb
26         dc.w $01b8,$0bab,$01ba,$0cbc,$01bc,$0dcd,$01be,$0eef
27
28 ;       Come vedete qua sono definiti tutti i 32 registri colore dell'Amiga,
29 ;       infatti ho caricato col KEFCON una figura a 32 colori e questa è
30 ;       la sua PALETTE generata assieme alla figura in RAW.
31
32 ;       Qua si possono fare eventuali effetti video con i WAIT....
33
34         dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

Questa copperlist è sufficiente per la visualizzazione di una figura. Procediamo dunque con il primo esempio di visualizzazione di un *playfield* con 3 bitplanes, ossia 8 colori. Nel primo esempio di questa lezione, visualizziamo una figura già convertita in formato RAW, ed è presente sul disco del corso: per "caricare" la figura in memoria esiste una direttiva dell'ASMONE, chiamata `INCBIN`, che permette appunto di caricare dal disco un certo file di dati e copiarlo nel punto del nostro programma dove sta l'`incbin`: se per esempio preparassimo una copperlist e la salvassimo come file, potremmo caricarcela così:

```
1  COPPERLIST:
2          incbin  "copper1"
```

Il risultato è lo stesso che mettere tanti dc.w equivalenti alle word che stanno nel file `copper1`. Nel nostro caso carichiamo l'immagine sotto una label `PIC`:

```
PIC:
        incbin   "amiga.320*256*3"
```

La figura comunque non è in formato TESTO, ma proprio in byte che compongono i bitplane: provate a caricarla in un buffer di testo e noterete che non si tratta di un testo.

Come notate il nome della figura è dato secondo le caratteristiche della figura stessa; è bene scegliere nomi per le figure che rispecchino le sue caratteristiche per non rischiare di dimenticarsi di che dimensione e quanti bitplanes hanno le figure RAW che convertite. Dalla lunghezza di questa immagine raw però si può dedurre la risoluzione e il numero di bitplanes: è lunga 30720 bytes, ossia 40*256*3 (40 bytes per linea*256 linee, moltiplicato 3 bitplanes fa 30720). Basterà dunque dire al COPPER che la figura si trova alla label `PIC`, e il gioco è fatto.

Comunque per puntare i bitplanes senza rischi di errore bisogna mettere i puntatori nella copperlist. Infatti i puntatori dei bitplanes possono contenere una word ciascuno, ossia la metà di un indirizzo (infatti un indirizzo è lungo una longword! es *$00020000*). Se usiamo il processore possiamo anche caricare 2 registri di una word con un solo `move.l`

```
        MOVE.L  #BITPLANE00,$dff0e0     ; BPL0PT
        MOVE.L  #BITPLANE01,$dff0e4     ; BPL1PT (2 word piùavanti di $dff0e0)
```

Ma nella copperlist come noto ogni move può essere di una WORD solamente:

```
        MOVE.W  #$123,$dff180   diventa   dc.w  $180,$123
```

Nel caso dei puntatori ai bitplanes allora dovremmo "spezzare" ogni indirizzo lungo una longword in 2 words per poter fare così:

```
        MOVE.W  #BITPL,$dff0e0  ; BPL0PTH (H=word ALTA dell'indirizzo)
        MOVE.W  #ANE00,$dff0e2  ; BPL0PTL (L=word BASSA dell'indirizzo)
        MOVE.W  #BITPL,$dff0e4  ; BPL1PTH
        MOVE.W  #ANE01,$dff0e6  ; BPL1PTL

BPLxPTH         = BitPLane x PoinTer High word , puntatore word alta
BPLxPTL         = BitPLane x PoinTer Low word , puntatore word bassa
```

Abbiamo spezzato `BITPLANE00` (lungo una longword) in 2 words `BITPL` e `ANE00`, e abbiamo ottenunto lo stesso risultato del `MOVE.L` con 2 `MOVE.W`, adatti alla copperlist, dove tradurremmo così:

```
        dc.w    $e0,BITPL       ; BPL0PTH \primo bitplane
        dc.w    $e2,ANE00       ; BPL0PTL /

        dc.w    $e4,BITPL       ; BPL1PTH \secondo bitplane
        dc.w    $e6,ANE01       ; BPL1PTL /

        (infatti $dff0e0 si traduce in copperlist in $e0, ecc.)
```

Questa divisione si dice *divisione di una longword in word alta e word bassa*, dove la *word alta* è quella a sinistra, `BITPL`, quella *bassa* è quella destra, qua `ANE00`. Facciamo un esempio con indirizzi veri:

Il `bitplane0` si trova a *$23400*, il `bitplane1` a *$25c00*

```
        dc.w    $e0,$0002       \primo bitplane (word alta)     \$00023400
        dc.w    $e2,$3400       /               (word bassa)    /

        dc.w    $e4,$0002       \secondo bitplane (word alta)   \$00025c00
        dc.w    $e6,$5c00       /                 (word bassa)  /
```

Vi starete già immaginando che per mettere gli indirizzi giusti nella copperlist sia necessario controllare a che indirizzo sta la figura e cambiare a mano le word in questione. Invece basta una piccola routine di una decina di istruzioni per risparmiarci il lavoro di "spezzare" gli indirizzi e

metterli nella copperlist al punto giusto. Questa routine si può usare per *puntare* qualsiasi figura di qualsiasi dimensione con il numero di bitplanes che vogliamo, basta cambiare i parametri!!! Il trucco sta nell'uso di una istruzione particolare del 68000, lo SWAP, che in inglese significa SCAMBIA, ed in effetti SCAMBIA le 2 words di una longwords facendo diventare BASSA quella ALTA e viceversa:

```
        MOVE.L  #CANETOPO,d0    ; in d0 mettiamo la longword CANETOPO

        SWAP    d0              ; SCAMBIAMO LE WORDS: il risultato è che
                                ; in d0 abbiamo TOPOCANE!!!!
```

Questo comando *funziona solo sui registri DATI*. Allo stesso modo $00023400 viene scambiato in $34000002. Vediamo la routine:

```
        MOVE.L  #PIC,d0         ; in d0 mettiamo l'indirizzo della PIC,
                                ; ossia dove inizia il primo bitplane

        LEA     BPLPOINTERS,A1  ; in a1 mettiamo l'indirizzo dei
                                ; puntatori ai planes della COPPERLIST
        MOVEQ   #2,D1           ; numero di bitplanes -1 (qua sono 3)
                                ; per eseguire il ciclo col DBRA
POINTBP:
        move.w  d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
                                ; nella word giusta nella copperlist
        swap    d0              ; scambia le 2 word di d0 (es: 1234 > 3412)
                                ; mettendo la word ALTA al posto di quella
                                ; BASSA, permettendone la copia col move.w!!
        move.w  d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
                                ; nella word giusta nella copperlist
        swap    d0              ; scambia le 2 word di d0 (es: 3412 > 1234)
                                ; rimettendo a posto l'indirizzo.
        ADD.L   #40*256,d0      ; Aggiungiamo 10240 ad D0, facendolo puntare
                                ; al secondo bitplane (si trova dopo il primo)
                                ; (cioè aggiungiamo la lunghezza di un plane)
                                ; Nei cicli seguenti al primo faremo puntare
                                ; al terzo, al quarto bitplane eccetera.

        addq.w  #8,a1           ; a1 ora contiene l'indirizzo dei prossimi
                                ; bplpointers nella copperlist da scrivere.
        dbra    d1,POINTBP      ; Rifai D1 volte POINTBP (D1=num of bitplanes)
```

dove cambiamo questa parte di copperlist:

```
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane (BPL0PT)
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane (BPL1PT)
        dc.w $e8,$0000,$ea,$0000        ;terzo   bitplane (BPL2PT)
```

La routine non fa altro che prendere l'indirizzo del bitplane, copiarne la word BASSA nella copperlist alla word dopo il $e2, ossia il puntatore della word bassa dell'indirizzo (che si trova 6 bytes dopo BPLPOINTERS, infatti viene usato un `move.w d0,6(a1)`, dove in a1 c'è l'indirizzo di BPLPOINTERS). Dopo aver sistemato la word bassa, con lo SWAP scambiamo word bassa con word alta, permettendo con un successivo `move.w d0,2(a1)` di copiare la word ALTA, anziché quella bassa, nella word dopo il $e0, ossia il puntatore della word alta del primo bitplane, che si trova, appunto, 2 bytes dopo BPLPOINTERS. A questo punto abbiamo PUNTATO il primo bitplane: (es. a $23400)

```
BPLPOINTERS:
        dc.w $00e0,$0002,$00e2,$3400    ; BPL0PT - primo   bitplane * PUNTATO *
        dc.w $00e4,$0000,$00e6,$0000    ; BPL1PT - secondo bitplane
        dc.w $00e8,$0000,$00ea,$0000    ; BPL2PT - terzo   bitplane

                    ^           ^
                    |           |
                  2(a1)       6(a1)     ; notare l'uso degli offset per
                                        ; inserire le word nel punto
                                        ; giusto.
```

NOTA: con il `move.w d0,x(a1)` copiamo la word bassa della long in `d0` perché la copia avviene in questo modo:

```
1       move.w  #CANETOPO,2(a1) ; in 2(a1) viene copiato TOPO
```

Successivamente rimettiamo a posto l'indirizzo con un altro `SWAP`, per poter passare al bitplane successivo con un `ADD.L #LUNGHEZZABITPLANE,d0`. Con un `addq.w #8,a1` passiamo a puntare il secondo bitplane, infatti se in `a1` c'era l'indirizzo di `BPLPOINTERS`, aggiungendo 8 bytes (4 words) passiamo in questo punto:

```
1  BPLPOINTERS:
2       dc.w $00e0,$0002,$00e2,$3400    ; BPL0PT - primo   bitplane * PUNTATO *
3  a1PUNTAQUA:
4       dc.w $00e4,$0000,$00e6,$0000    ; BPL1PT - secondo bitplane
5       dc.w $00e8,$0000,$00ea,$0000    ; BPL2PT - terzo   bitplane
```

Ripetiamo quindi con un loop `DBRA d1,label` questa routine, in questo caso 3 volte, per puntare 3 bitplanes. (Come ricorderete al loop `DBRA` bisogna mettere nel registro che conta i cicli il numero richiesto meno 1, che è il primo che non viene contato: qua infatti in `d1` viene messo il valore 2.

Questa routine ha la struttura classica delle routines che generano effetti col copper, dunque capirla esattamente è fondamentale. Una routine simile la avete già trovata in `Lezione3h.s`, in quel caso c'erano dei loop `DBRA` per cambiare 29 wait della copperlist.

Caricate `Lezione4a.s` per vedere in pratica l'esecuzione di questa routine `PUNTABITPLANES` con il DEBUG.

A questo punto mancano solo due "rifiniture" al nostro listato per evitare problemi nel visualizzare un'immagine: un paio di istruzioni per disabilitare il chipset AGA, che rende compatibile con l'Amiga 1200 e 4000 il nostro lavoro, e qualche altra linea nella copperlist per far sparire gli sprites, che altrimenti girovagherebbero a caso per il nostro disegno provocando disturbi intermittenti. Per disabilitare l'AGA bastano queste 2 linee:

```
1       move.w  #0,$dff1fc              ; FMODE - Disattiva l'AGA
2       move.w  #$c00,$dff106           ; BPLCON3 - Disattiva l'AGA
```

E se volete proprio essere sicuri potete aggiungere questa: (palette sprite)

```
1       move.w  #$11,$dff10c            ; BPLCON4 - Resetta palette Sprite
```

Basta eseguirle dopo aver puntato la nuova copperlist, mentre per fermare gli sprites ubriachi basta puntare i loro puntatori a ZERO:

```
1       dc.w    $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000
2       dc.w    $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
3       dc.w    $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
4       dc.w    $13e,$0000
```

(Nota: I registri da `$dff120` a `$dff13e` si chiamano `SPR0PT`, `SPR1PT...SPR7PT`.)

Parleremo degli sprites in seguito, per ora basta che li togliate di mezzo aggiungendo con un semplice *<COPY+INSERT>* (*<Amiga+b+c+i>*) questo pezzo di testo alla vostra copperlist. Gli sprites non apparivano con i bitplanes azzerati, mentre accendendo anche un solo bitplane gli sprite si fanno vivi.

Finalmente potete vedere in pratica come viene visualizzata una figura caricandovi `Lezione4b.s` in un buffer di testo Fx a piacere.

Avete provato ad aggiungere effetti copper alla figura??? Caricate l'esempio `Lezione4c.s` per una fusione di alcuni degli effetti già visti.

Avete capito ora l'importanza del `WAIT` in una copperlist con dei bitplane? Ci può far cambiare i colori (e non solo) anche ogni linea! Non ci resta che visualizzare un'immagine fatta da voi anziché quella nel corso. Per fare ciò dovete disporre di una figura in 320x256 a 8 colori, se non la avete fatevela anche sommariamente con un programma di disegno, oppure convertite

con ADPRO o altro programma una figura a più colori in quel formato. Disponendo di quella immagine in formato IFF (su un disco formattato) con il nome che preferite, immaginiamo che si chiami "FIGURA", bisogna convertirla in formato RAW, ossia nel formato REALE dei bitplanes leggibili dal Copper, caricandola con l'iffconverter in questo disco, il "KEFCON", che ha molte funzioni di cui però parleremo in seguito. Leggete queste istruzioni prima di caricarlo: Il converter è programmato in assembler hardware, dunque non supporta il multitasking e non si può "spostare" la sua finestra per leggere la lezione, dato che la sua finestra è una copperlist propria e non di sistema, che però è compatibile AGA e non crea problemi (I buoni vecchi programmatori!). Preparatevi prima la figura in un disco formattato, che poi inserirete dopo aver caricato il KEFCON nel `df0:` (disk drive interno) o nel `df1:` (disk drive esterno) se lo avete. Una volta caricato, vi apparirà un quadro comandi in alto, con delle opzioni, quelle che ci interessano sono: (vi faccio uno schema dei "PULSANTI")

```
 ------          ----------
| SAVE |        | IFF ILBM |
 ------          ----------

 ------          ----------
| LOAD |        | READ DIR |
 ------          ----------

 ------
| QUIT |
 ------

 -----------------------------------
| QUA C'E UNA FINESTRELLA A STRISCIA | -Dove si deve scrivere il nome del file
 -----------------------------------
```

LOAD, SAVE e QUIT significano naturalmente CARICA, SALVA ed ESCI DAL PROGRAMMA READ DIR serve a visualizzare nella finestra a destra la lista dei file che sono nel disco, ossia la sua directory. IFF ILBM è un pulsante che indica il tipo di file che si può salvare o caricare, in questo caso è giustamente ad IFF ILBM perchè dobbiamo caricare una figura iff; successivamente per salvarla in RAW basterà clickare su quel pulsante, il quale si cambierà in "RAW NORM", e la figura sarà salvata come RAW; tra i possibili formati ci sono anche SPRITE e RAW BLIT, quelli li useremo in seguito. Per ora ci interessano solo "RAW NORM" e "COPPER" dove con COPPER si salva la PALETTE dei colori direttamente in un file di testo con i DC.W da inserire nella nostra copperlist!

Per fare la conversione, clickate sulla finestrella a striscia in basso a sinistra dove appare la scritta "allocated GFX-buffer" e si cambierà in `df0:` ossia il drive interno. Se avete inserito il disco con la figura iff in `df0:` lasciatelo, oppure indicate il nome del drive dove avete messo il disco (ad esempio `df1:`, oppure `dh0:` per l'hardisk). Per leggere la lista dei file premete READ DIR, e potrete selezionare la vostra figura clickando col mouse sopra il suo nome e premendo il pulsante "LOAD". Vedrete a questo punto comparire la figura, che potete vedere per intero scorrendola in alto e basso con i tasti cursore. Notate che una volta caricata la figura appaiono le sue caratteristiche nella finestrella a striscia: "bitplane $2800, total $7800". Infatti ogni singolo bitplane è lungo $2800 (ossia 10240 decimale, 40*256), e in totale la figura RAW è lunga $7800, ossia 30720 (40*256*3). Sopra sono indicate anche le altre caratteristiche:

```
WIDTH: 320 (larghezza), HEIGHT 256 (LUNGHEZZA), DEPTH 3 (NUMERO BITPLANES)
```

Ora clickate sul pulsante "IFF ILBM" facendolo cambiare in "RAW NORM"; per salvare la pic in RAW premete un'altra volta col tasto sinistro sulla finestrella a striscia e definite il nome, ad

esempio “df0:FIGURA.RAW” e premete il pulsante “SAVE”. Il RAW da includere con l’INCBIN è salvato! Ora manca solo la palette dei colori per la nostra COPPERLIST; per selezionare il modo ’salvataggio palette clickate 5 volte sul gadget “RAW NORM”, ossia fino a che non si cambia in “COPPER”. Per salvarla ripetete l’operazione di clickaggio della finestrella a striscia, dategli un nome, es: “df0:FIGURA.S”, e premete il pulsante “SAVE”. A questo punto potete anche uscire, perché abbiamo salvato sia il .RAW che il testo della palette in dc.w da includere nella nostra copperlist.

Per visualizzarvi la figura, caricate l’esempio `Lezione4b.s` e fate queste 2 sostituzioni: cambiate il nome della figura da caricare mettendoci la vostra, e inserite la palette della vostra figura eliminando quella preesistente:

```
PIC:
        incbin  "amiga.320*256*3"
```

lo potete cambiare in:

```
PIC:
        incbin  "df0:figura.raw"
```

oppure, scrivendo *<v df0:>* dalla linea comandi basta un:

```
PIC:
        incbin  "figura.raw"
```

Per la palette potete fare in 2 modi: o caricate `FIGURA.S` in un altro buffer e poi lo copiate con *<Amiga+b+c+i>*, oppure potete usare il comando dell’Asmone *<I>*, ossia INSERT, che inserisce un testo nel punto in cui vi trovate col cursore prima di premere ESC per passare alla linea comandi. Comunque facciate, togliete la vecchia palette con il CUT (TAGLIA) dell’editor: *<Amiga+b>* per selezionare il blocco, *<Amiga+x>* per eliminarlo.

Ha funzionato?? Spero di si, altrimenti significa che avete sbagliato qualche passaggio, pena la ripetizione di tutta l’operazione.

Per continuare in bellezza ora proviamo a visualizzare una figura a 32 colori: basta che disponiate della consueta figura in 320x256, questa volta a 32 colori (se proprio non la avete caricate il DeLuxe Paint e “schizzate” qualche oscenità). Convertitela come avete fatto per la figura precedente, questa volta noterete che dopo il caricamento le caratteristiche cincideranno con le previsioni: “bitplane $2800, total $c800”, infatti ogni bitplane è sempre di $2800 bytes, mentre il totale è $c800, ossia 51200 ($2800*5), essendo una figura a 32 colori composta di 5 bitplanes. Salvate il .RAW e la palette, ad esempio con nomi come `FIGURA32.RAW` e `FIGURA32.s`.

Per visualizzarla dovrete fare al sorgente i due cambiamenti di prima, ossia far caricare la nuova figura dall’incbin e sostituire la vecchia palette con quella nuova (che come noterete è più lunga coinvolgendo tutti i 32 registri colore). In più dovete cambiare il numero di bitplanes nella routine di puntamento e aggiungere i bitplanes mancanti alla copperlist:

```
        MOVE.L  #PIC,d0         ; in d0 mettiamo l'indirizzo della PIC,
        LEA     BPLPOINTERS,A1  ; in a1 i puntatori in COP
**->    MOVEQ   #4,D1           ; numero di bitplanes -1 (qua sono 5!!!!!)
                                ; per eseguire il ciclo col DBRA
POINTBP:
        ....
```

Cambiate il `MOVEQ #2,d1` in `MOVEQ #4,d1`, ossia mentre prima eseguivamo 3 cicli `DBRA` per 3 bitplanes (3-1=2), ora eseguiamo 5 cicli (5-1=4) per 5 bitplanes. Ma allora bisogna aggiungere i puntatori di bitplanes mancanti nella copperlist:

```
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo   bitplane - BPL2PT
        dc.w $ec,$0000,$ee,$0000        ;quarto  bitplane (AGGIUNTO ORA!)
        dc.w $f0,$0000,$f2,$0000        ;quinto  bitplane (AGGIUNTO ORA!)
```

Ultima e più importante modifica bisogna “accendere” 5 bitplanes anziché 3, questo lo si può fare modificando il `$dff100` (`bplcon0`) nella copperlist:

```
                    ; 5432109876543210
        dc.w    $100,%0101001000000000  ; bits 14,12 accesi!! (5 = %101)
```

Assemblando il tutto dovrebbe apparire la figura a 32 colori.

Da questi due esempi potete intuire facilmente come visualizzare immagini a 2,4 e 16 colori! basta cambiare il numero di loop nella routine che punta i bitplanes e sistemare i bit giusti in `$dff100` (`BPLCON0`).

Vediamo ora di visualizzare una figura in EHB a 64 colori e una in HAM a 4096 colori, attivando i due modi grafici speciali.

Partiamo da quella in HAM: fatevi una figura in 320x256 in HAM, oppure cercate una delle tante figure ham che si trovano spesso nei dischi di riviste o nei dischi di immagini “SEXY”, che sono per lo più in HAM, dato che la fedeltà è importantissima per l’immagine di una donna nuda. Credo anzi che sia più piacevole visualizzare una donna nuda che un cesto di frutta. Caricate come da copione la figura in HAM 320x256 col KEFCON e salvate il RAW e la COPPERLIST. Purtroppo il KEFCON ha un errore di programmazione per cui quando vengono caricate su A4000 figure a 6 bitplanes, siano HAM o EHB, si provoca una specie di “spappolamento” dei numeri e dei segni di punteggiatura (,.:) del quadro di comando, (sull’Amiga500/2000/600 funziona invece!) per cui potete vedere correttamente solo le parole, ma non è certo un problema, in quanto basta clickare sulla finestrella a striscia ed aggiungere un .RAW, per esempio, al nome della PIC che è visibile, poi un .s per salvare la copperlist. Le modifiche da fare sono l’aggiunta degli ultimi puntatori al bitplane 6 in copperlist, la sostituzione della palette e il settaggio del numero di cicli della routine di puntamento a 6:

```
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo   bitplane - BPL2PT
        dc.w $ec,$0000,$ee,$0000        ;quarto  bitplane - BPL3PT
        dc.w $f0,$0000,$f2,$0000        ;quinto  bitplane - BPL4PT
        dc.w $f4,$0000,$f6,$0000        ;sesto   bitplane (AGGIUNTO ORA!)


**->    MOVEQ   #5,D1           ; numero di bitplanes -1 (qua sono 6!!!!!)
                                ; per eseguire il ciclo col DBRA
POINTBP:
        ...

Nonchè il BPLCON0:

          ; 5432109876543210
 dc.w $100,%0110101000000000    ; ---> 6 plane in HAM lowres (4096 colori)
                                ; BIT 11 settato = HAM!
```

Il funzionamento teorico dell’HAM sarà affrontato meglio in seguito.

Per visualizzare una immagine in Extra Half Bright, invece, convertitene una col KEFCON, fatela caricare dall’incbin, sostituite la palette, lasciate puntare 6 bitplanes alla routine e azzerate il bit 11 del `bplcon0`:

```
          ; 5432109876543210
 dc.w $100,%0110001000000000    ; ---> 6 plane in EHB lowres (64 colori)
                                ; BIT 11 azzerato = Extra Half Bright
```

NOTA: Il modo EHB ha 64 colori, ma non sono tutti e 64 selezionabili liberamente, in quanto l’Amiga ha solo 32 registri colore; gli altri 32 colori sono come i primi 32 ma più scuri, a “mezza luce”, ossia *Half Bright*.

Ora che sappiamo visualizzare delle figure, vediamo che effetti si possono fare con i registri di scorrimento. Caricate la `LEZIONE5.TXT` con *<r>*

CAPITOLO 5

LEZIONE 5 - LO SCROLLING ORIZZONTALE E VERTICALE

In questa lezione tratteremo lo scorrimento orizzontale e verticale delle figure, nonché alcuni effetti speciali.

Cominciamo dallo scorrimento orizzontale: l'Amiga ha un registro speciale dedicato allo scorrimento, il `BPLCON1` (`$dff102`), che può far scorrere di un pixel alla volta verso destra i bitplane per un massimo di 15 pixel. Ciò è ottenuto dal copper ritardando il trasferimento dei dati dei bitplane, che arrivano "dopo" di uno o più pixel. Si possono inoltre scorrere separatamente i bitplanes pari e quelli dispari: i bitplanes dispari sono chiamati *PLAYFIELD 1* (1, 3, 5), mentre quelli pari *PLAYFIELD 2* (2, 4, 6). Il `$dff102`, lungo una word, è diviso in 2 byte: quello alto, ossia quello a sinistra, (`$xx00`), composto dai bit dal 15 all'8, non è utilizzato e bisogna lasciarlo a zero, mentre il byte basso (`$00xx`) controlla lo scroll:

```
$dff102, BPLCON1 - Bit Plane Control Register 1

BITS            NOME-FUNZIONE

15      -       X
14      -       X
13      -       X
12      -       X
11      -       X
10      -       X
09      -       X
08      -       X
07      -       PF2H3   \
06      -       PF2H2    \ 4 bit per scroll PLANES PARI (playfield 2)
05      -       PF2H1    /
04      -       PF2H0   /
03      -       PF1H3   \
02      -       PF1H2    \4 bit per scroll PLANES DISPARI (playfield 1)
01      -       PF1H1    /
00      -       PF1H0   /
```

In pratica si deve agire sulla word in maniera simile ai registri colore: mentre nei registri colore si agisce su 3 componenti RGB, che vanno da 0 a 15, ossia da 0 a $F, qua agiamo su 2 sole

componenti che vanno da $0 a $f, come il GREEN e BLU del $dff180 (COLOR0):

```
        dc.w    $102,$00XY      ; BPLCON1 - dove: X= scroll bitplanes PARI
                                ;                 Y= scroll bitplanes DISPARI
```

Alcuni esempi: (per la Copperlist)

```
        dc.w    $102,$0000      ; BPLCON1 - scroll zero, posizione normale
        dc.w    $102,$0011      ; BPLCON1 - scroll = 1 in entrambi i playfield,
                                ; ossia in tutta la figura
        dc.w    $102,$0055      ; BPLCON1 - scroll = 5 per tutta la figura
        dc.w    $102,$00FF      ; "" scroll al massimo (15) per tutta la figura
        dc.w    $102,$0030      ; "" scroll = 3 solo per i bitplanes PARI
        dc.w    $102,$00b0      ; "" scroll = $B solo per i bitplanes DISPARI
        dc.w    $102,$003e      ; "" scroll = 3 per i bitplanes PARI e di $e
                                ; per i bitplanes dispari
```

niente di più facile! Basta cambiare il valore di scroll ogni FRAME per creare uno scorrimento dell'intero schermo con un solo MOVE!!!

Caricate l'esempio Lezione5a.s per vedere in pratica il funzionamento.

In questo esempio il $dff102 (BPLCON1) viene cambiato all'inizio della COPPERLIST, dunque si muove tutta la figura. è possibile mettere molti $dff102 (BPLCON1) a varie linee dello schermo con la tecnica dei WAIT: nell'Esempio Lezione5b.s ce ne sono due, che fanno scorrere separatamente la scritta "COMMODORE" e "AMIGA". Mettendo un $dff102 (BPLCON1) per linea con i WAIT, si possono fare i noti effetti di ondeggiamento delle figure.

Vediamo ora lo scorrimento verticale. Il modo più semplice per fare questo scorrimento è di puntare più in alto o più in basso nella figura i puntatori ai bitplanes in copperlist, in modo che la figura sembri più "alta" o più "bassa". Immaginiamo di vedere una immagine attraverso un foro rettangolare, una specie di finestra (il video):

```
 ---------------
|               | 1
|               | 2
|     AMIGA     | 3
|               | 4
|               | 5
 ---------------
```

Vediamo in questo caso la scritta AMIGA al centro della finestra, e abbiamo puntato i bitplane alla linea 1 (cioè lo schermo inizia con la linea 1, per cui AMIGA si trova alla linea 3). Se puntiamo lo schermo alla linea 2, cosa succede???

```
 ---------------
|               | 2
|     AMIGA     | 3
|               | 4
|               | 5
|               | 6
 ---------------
```

Succede che AMIGA "sale" perché la finestra (il video) scende, ossia punta più in basso nella figura. Essendo il moto relativo, se vediamo dal finestrino di un treno in corsa un albero che si sposta, in realtà l'albero è "fermo" e noi ci spostiamo. Qua succede una cosa analoga. Ma per far andare in alto o in basso una figura, quanto dobbiamo aggiungere o diminuire ai puntatori dei bitplanes??? I byte di una linea. Ossia 40 per una figura in LOW RES 320x256 e 80 per una figura in HIGH RES 640x256, infatti esaminiamo questo caso:

```
1234567890
..........
```

```
....++....
...+..+...
...++++...
...+..+...
...+..+...
..........
```

Abbiamo un ipotetico bitplane con 10 byte per linea, che può essere a zero (.) o ad 1 (+), in questo caso raffigura una "A". Per spostare la "A" in alto, dobbiamo "puntare" una linea più in basso, ossia 10 bytes più in basso, e per puntare più in basso, occorre AGGIUNGERE 10 (`add.l #10,puntatori`)

```
1234567890
....++....
...+..+...
...++++...
...+..+...
...+..+...
..........
..........
```

Allo stesso modo, per farla "scendere", dobbiamo puntare una linea più in alto, ossia 10 bytes più in alto (`SUB.L #10,puntatori`):

```
1234567890
..........
..........
....++....
...+..+...
...++++...
...+..+...
...+..+...
```

In pratica per fare questo dobbiamo ricordarci che i puntatori in copperlist hanno l'indirizzo dei plane (che noi cambieremo) divisi in 2 word. Il problema è facilmente risolvibile con una lieve modifica della routine di puntamento dei bitplane, infatti dobbiamo "prendere" l'indirizzo dei bitplanes dalla copperlist (operazione contraria), aggiungere o sottrarre 40 per lo scroll, e rimettere il nuovo indirizzo nella copperlist con la vecchia routine di puntamento. Vedetevi l'esempio `Lezione5c.s` che usa questo sistema.

Ora caricatevi l'esempio `Lezione5d.s`, in cui sono presenti le due routines di scroll orizzontale e verticale contemporaneamente.

In `Lezione5d2.s` troverete un'altra applicazione dello scroll orizzontale insieme al `$dff102` (`bplcon1`), ossia la distorsione in movimento.

Vedremo ora i registri più importanti per gli effetti speciali video Amiga, ossia i *moduli*: `$dff108` e `$dff10a` (`BPL1MOD` e `BPL2MOD`). Ci sono due registri modulo perché si può cambiare il modulo separatamente per i bitplanes pari e per quelli dispari, come il `BPLCON1` (`$dff102`). Per operare sulla nostra figura a 3 bitplanes dovremo agire su entrambi i registri. Avrete notato che quando una immagine in LOW RES 320x256 viene visualizzata, il pennello va a capo ogni 40 bytes, mentre i dati sono tutti di seguito. Allo stesso modo, nel caso di una figura in HI-RES 640x256 il pennello va a capo ogni 80 bytes. Infatti il modulo viene automaticamente assegnato quando si setta il `$dff100` (`BPLCON0`): se viene selezionato il LOWRES il copper sa che una figura in lowres ha 40 bytes per linea, dunque partendo a visualizzare dall'inizio dello schermo (in alto a sinistra), si legge 40 bytes e scrive col pennello elettronico la prima linea, poi "va a capo" e i dati che seguono li scrive alla linea dopo, e così via. La figura in memoria però ha i dati tutti

consecutivi, non c'è una figura "quadrata" ! La memoria è una fila di byte consecutivi, per cui ogni bitplane è una linea consecutiva di dati: immaginate di dividere le 256 linee dello schermo, lunghe 40 bytes ciascuna, e di metterle l'una dopo l'altra per fare una sola linea di 40*256 bytes, ottenendone una lunga una settantina di metri: questa sarebbe la linea come è veramente in memoria. Mettendo il modulo a zero, come abbiamo fatto fino ad ora, lasciamo andare "a capo" come il LOWRES o HIGHRES comanda, ossia ogni 40 o 80 linee, e la visualizzazione è normale. Il valore che mettiamo al modulo viene *addizionato* ai puntatori ai bitplanes alla *fine* della linea, ossia una volta raggiunto il byte 40. In questo modo possiamo *saltare* dei bytes, che non vengono visualizzati. Per esempio se aggiungiamo 40 ad ogni termine di linea ne saltiamo una intera, per cui ne viene visualizzata una ogni due, infatti:

```
- IMMAGINE NORMALE -

....................    ; al termine di questa linea "salto" 40 bytes
.........+..........
........+++.........    ; e visualizzo questa linea, poi "salto"...
.......+++++........
......+++++++.......    ; e visualizzo questa linea, poi "salto"...
.......+++++........
........+++.........    ; e visualizzo questa linea, poi "salto"...
.........+..........
....................    ; e visualizzo questa linea, poi "salto"...
```

Il risultato sarà che visualizziamo solo una linea ogni due:

```
- IMMAGINE MODULO 40 -

....................    ; al termine di questa linea "salto" 40 bytes
........+++.........    ; e visualizzo questa linea, poi "salto"...
......+++++++.......    ; e visualizzo questa linea, poi "salto"...
........+++.........    ; e visualizzo questa linea, poi "salto"...
....................    ; e visualizzo questa linea, poi "salto"...
....................
....................
....................
....................
```

La figura apparirà schiacciata, lunga la metà, inoltre andremo a visualizzare anche byte "sotto" la nostra figura, dato che lo schermo finisce sempre alla linea 256: in pratica visualizziamo sempre 256 linee, ma in un raggio di 512 linee di cui visualizziamo solo una linea ogni due. Provate a ricaricare `Lezione5b.s` e modificate i moduli nella copperlist:

```
1       dc.w    $108,40         ; Bpl1Mod
2       dc.w    $10a,40         ; Bpl2Mod
```

Noterete che l'immagine è alta la metà come previsto e la parte inferiore dello schermo è riempita dai bitplane che "avanzano", ossia dal secondo bitplane visualizzato sotto il primo, e dal terzo visualizzato sotto il secondo mentre dopo il terzo si vede la memoria dopo la figura, insomma vengono visualizzate 256 linee in un raggio di 512. Provate a saltare 2 linee, saltando 80 bytes ogni 40 visualizzati:

```
1       dc.w    $108,40*2       ; Bpl1Mod
2       dc.w    $10a,40*2       ; Bpl2Mod
```

La figura si dimezzerà ancora, e spunteranno in basso altri bytes. Verificherete un dimezzamento dell'altezza continuando con moduli di 40*3, 40*4,40*5 eccetera, fino a rendere illeggibile il disegno. Se scegliete un modulo che non sia multiplo di 40 causerete lo "sfaldamento"

dell'immagine, infatti il copper visualizzerà le linee partendo non dal loro inizio ma da una parte sempre diversa.

Vedetevi `Lezione5e.s` per una veloce routine che aggiunge 40 al modulo per dimezzare la figura.

I moduli oltre che positivi possono essere anche negativi. In questo caso viene sottratto il numero negativo in questione alla fine di ogni linea visualizzata. In questo caso si possono creare effetti strani: immaginatevi di mettere il modulo come -40: in questo caso, il copper legge 40 bytes, li visualizza in una linea, poi torna indietro di 40 bytes, visualizza gli stessi dati nella linea successiva, poi torna indietro di 40 bytes, e così via. In pratica non avanza oltre i primi 40 bytes e ogni linea ricopia la prima linea: se per esempio abbiamo la prima linea tutta nera, le altre riprodurranno questa e lo schermo sarà tutto nero. Se ci fosse un solo punto nel mezzo della linea, questo sarebbe ridisegnato ogni linea e si produrrebbe una riga verticale:

```
.........+........     ; linea 1 (sempre ridisegnata: modulo -40!)
.........+........     ; linea 2
.........+........     ; linea 3
.........+........     ; linea 4
.........+........     ; linea 5
.........+........     ; linea 6
.........+........     ; linea 7
.........+........     ; linea 8
.........+........     ; linea 9
.........+........     ; linea 10
```

Allo stesso modo ogni colore provoca una specie di "colatura" fino alla fine dello schermo. Questo effetto è stato usato in giochi come Full Contact, nel red-sector demomaker e in moltissimi altri programmi.

Vediamo il suo funzionamento in pratica in `Lezione5f.s`.

Suggestivo e semplice da fare, o sbaglio? è detto anche effetto *flood*. Il modulo viene addizionato, ogni fine linea, ai puntatori dei bitplanes che "camminano" nella memoria per visualizzare tutta la figura. Quindi addizionando un numero negativo, sottraiamo. In questo caso specifico, i puntatori dopo aver trasferito ogni linea assumono il valore X+40, vengono quindi incrementati del valore del modulo (=-40: la lunghezza in byte di una singola riga di bitplane, al negativo): decrementati dunque di '40' byte, assumono infine di nuovo il valore X di partenza.

```
+---->->->--------+
|                 |
|BPL POINTER=  X+ 0.....................................39
|                 |                                     |
|INIZIO RIGA -+---xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx---+- ULTIMO BYTE ->
|     (X)     |   |                                     |   |     (X+39)
|              +---+                                     +---+
|                 |
| RIGA DOPO -+----xxxx[...]
^            |    |
|            +-X+ 40  (il puntatore, dopo il trasferimento, ha camminato per
^                 |   l'intera lunghezza della riga (40 byte), fermandosi
|                 |   al 40esimo che altri non è se non il primo byte della
^                 |   riga successiva)
|                 +-> (Qui viene ADDIZIONATO al puntatore di ogni plane il
|                 |   valore del modulo a lui assegnato: in questo caso '-40')
|                 +-> X=X+(-40) => X=X-40 => X=0 >-+
|                 |                                |
+----------<-<-<--+------------<-<-<---------------+
```

Visto? Il puntatore, sul più bello, arrivato a X+40, viene sottratto di 40 e torna all'inizio della riga appena trasferita, visualizzando ancora la stessa riga in quella sotto, in quanto il pennello

elettronico cammina sempre verso il basso e disegna quanto gli viene "detto" al punto in cui si trova, in questo caso sempre la stessa riga, ripetuta.

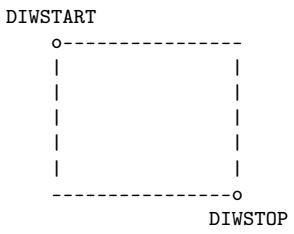
Abbiamo visto in Lezione5f.s anche l'effetto specchio, ossia il modulo -80. Vediamolo da solo nell'esempio Lezione5g.s.

Vediamo ora come utilizzare molti $dff102 (BPLCON1) consecutivi in copperlist per creare un effetto di ondulazione: caricate Lezione5h.s

Vediamo un utilizzo particolare dello scroll con i bitplanes: Lezione5i.s è un cosiddetto GRAPHIC-SCANNER, un antenato dei GFX-RIPPERS, ossia i programmi che "RUBANO" figure dalla memoria. Questo breve programmino serve semplicemente a mostrare la memoria CHIP, con tutte le figure visibili in essa contenute.

Ancora un esempio inerente ai moduli in Lezione5l.s, questa volta per fare un "allungamento" della figura anziché un dimezzamento.

In Lezione5m.s vedremo un'altro metodo per spostare in basso e in alto la figura, questa volta modificando il DIWSTART ($dff08e) I registri DIWSTART e DIWSTOP determinano l'inizio e la fine della "finestra video", ossia la parte rettangolare di schermo dove vengono visualizzati i bitplanes. DIWSTART contiene le coordinate YYXX dell'angolo in alto a sinistra, dove inizia il "rettangolo video", mentre DIWSTOP contiene le coordinate dell'angolo in basso a destra:



In questi registri però non si possono indicare tutte le possibili coordinate XX e YY, infatti sia la posizione XX che YY sono byte, e come sappiamo i byte possono raggiungere 256 diversi valori ($00-$ff). Vediamo in quali posizioni possiamo cominciare la finestra video col DiwStart e in quali possiamo terminarlo col DiwStop.

```
1       dc.w    $8e,$2c81       ; DiwStrt YY=$2c,       XX=$81
2       dc.w    $90,$2cc1       ; DiwStop YY=$2c(+$ff), XX=$c1(+$ff)
```

La finestra video normale ha questi valori di DIWSTRT e DIWSTOP; la posizione verticale, la YY, funziona esattamente come la posizione YY dei wait del copper: infatti se col copper aspettate una linea sopra $2c e ci fate delle sfumature, non saranno visibili perché troppo in alto, o comunque risulterà sopra qualsiasi figura visibile; analogamente al wait dopo la linea $FF la posizione riparte da $00, che sarebbe $FF+1. infatti lo schermo inizia dalla posizione verticale $2c, e finisce al $2c dopo la linea 256, ossia $FF+$2c, ossia $12b, visualizzando un totale di 256 linee, come previsto. Per esempio per uno schermo alto 200 linee dovremo mettere questo DiwStop:

```
1       dc.w    $90,$f4c1       ; DiwStop YY=$2c(+$ff), XX=$f4
```

Infatti $f4-$2c = 200. Se indichiamo $00,$01...aspetteremo dopo la linea $ff. Le limitazioni sono queste: il DiwStart può posizionarsi verticalmente in una delle posizioni YY da $00 a $FF, ossia fino alla linea 200. La finestra video dunque non può comunicare dalla linea 201 o seguenti, sempre prima. Per il DIWSTOP i progettisti si sono serviti di uno stratagemma: se il valore YY è sotto $80, ossia 128, allora aspetta le linee sotto $FF, per cui il $2c si riferisce a $2c+$FF, ossia la linea 256. Se il numero è superiore a $80 allora lo prende così com'è, (dato che non esistono linee $80+$ff=383!!), e aspetta veramente le linee 129,130 eccetera. Dunque, se il DIWSTART può arrivare al massimo alla linea $FF partendo dallo ZERO, il DIWSTOP può superare

la linea $FF e arrivare ai limiti del video in basso, ma non può partire da linee inferiori alla $80. Questo trucco è stato fatto considerando i numeri con il bit 7 a zero (quelli, appunto, prima $80), come se avessero un ipotetico bit 8 impostato, il che aumenta tutto di $FF. Quando invece il bit 7 viene impostato (i numeri dopo $80 lo hanno impostato) allora il bit fantasma sparisce e i numeri sono presi per quello che sono. Per quanto riguarda la linea orizzontale il diwstart può partire da una qualsiasi XX da $00 a $FF, quindi fino alla posizione 256, (ricordatevi però che lo schermo parte dalla posizione $81 e non da $00, dunque è la posizione 126 dall'inizio dello schermo!). Il DiwStop invece con $00 indica la linea 127, e proseguendo può raggiungere la fine del borso destro dello schermo, infatti ha il bit 8 "fantasma" sempre ad 1, per cui viene sempre aggiunto $FF al suo valore di XX. In definitiva il DiwStart può posizionarsi in una qualsiasi delle posizioni XX e YY da $00 a $FF, mentre il DiwStop può posizionarsi orizzontalmente dopo la linea $FF, e verticalmente dalla linea $80 alla linea $FF, dopodiché i numeri da $00 a $7f sono, come nel wait dopo la linea $FF, le linee 201,202 eccetera, per cui $2c è $2c+$ff.

In Lezione5m2.s, Lezione5m3.s e Lezione5m4.s viene trattato questo argomento.

Come termine della LEZIONE5, caricatevi Lezione5n.s, che è un riassunto delle lezioni precedenti e in più è il primo listato che suona anche la musica.

Una volta capito questo esempio, non vi resta che caricare la LEZIONE6.TXT.

CAPITOLO 6

LEZIONE 6 - I FONTS E LO SCROLLING

In questa lezione vedremo come visualizzare dei testi sullo schermo, come far scorrere schermi più grandi della finestra video, e l'uso delle tabelle di valori predefiniti per simulare movimenti di rimbalzo e ondeggiamento.

Imparare a visualizzare delle scritte sullo schermo è importantissimo, non si può fare a meno di una routine di stampa caratteri in un gioco o in una demo grafica: se vogliamo scrivere il punteggio e il numero delle vite, o un messaggio tra un livello e l'altro, oppure il dialogo tra i personaggi, una scritta con i saluti agli amici, eccetera. è chiaro che non vengono visualizzate delle figure 320x256 con le scritte già fatte! Immaginatevi di voler visualizzare 5 pagine di testo per introdurre la storia del vostro gioco: "un cavaliere di un periodo storico imprecisato decise di andare alla ricerca del santo graal..." eccetera. Le soluzioni sono due: o vi disegnate col programma da disegno cinque figure col testo stampato, e in questo caso avremmo 5 figure da 40*256 = 51200 byte utilizzati, che vi rubano spazio su disco e memoria, oppure con 1k di FONT caratteri e pochi byte di routine che stampa quei caratteri fate lo stesso lavoro, risparmiando 50k. Avrete presenti i FONT di caratteri del sistema operativo: TOPAZ, DIAMOND eccetera, che potete scegliere? Ebbene a noi non interessano i FONT di sistema, perché ne usiamo di nostri. Si possono usare anche i font di sistema, ma sono limitati, mentre facendosi i font e la routine che stampa i caratteri di quel font si possono visualizzare scritte di qualsiasi dimensione, anche colorate, basta disegnare il font e farsi la routine giusta. Una volta capito il sistema di *print*, ossia di *stampa* dei caratteri si possono fare variazioni senza difficoltà. Per cominciare vediamo come stampare un font piccolo, largo 8 pixel e alto 8, ad un solo colore. Come prima cosa bisogna disporre di un *bitplane* dove stampare il testo e di un *font caratteri* dove sono disegnati tutti i caratteri da copiare. Per il bitplane non ci sono problemi, infatti basta crearsi nel listato un pezzo di memoria azzerato della dimensione di un bitplane, e "puntarlo", ossia farlo visualizzare. Per fare uno spazio azzerato si può usare il comando `DCB.B 40*256,0` che, appunto, crea uno spazio azzerato della dimensione giusta; ma esiste una `SECTION` specifica per i *buffer azzerati*: la `section BSS`, in cui si può usare la sola direttiva `DS.B/DS.w/DS.l`, che stabilisce quanti bytes/word/longword azzerati creare. Il vantaggio sta nella lunghezza finale del FILE ESEGUIBILE: mentre creando lo spazio azzerato con un: `BITPLANE: dcb.b 40*256,0` i 10240 bytes sono aggiunti alla lunghezza totale del file, definendo una `Section BSS`:

```
        SECTION UnBitplaneQua,BSS_C     ; _C significa che deve essere caricata
                                        ; in CHIP RAM, senza il _C verrebbe
                                        ; caricata dove capita, anche in FAST!
                                        ; ma i bitplane devono essere in CHIP.
BITPLANE:
        ds.b    40*256          ; 10240 bytes a zero
```

Al file verrà aggiunto un *Hunk* di pochi bytes che "varrà" 40*256 bytes al momento del caricamento in memoria del file. Il `dcb.b 40*256,0` è come avere un ingombrante sacchetto di monete da 100 lire, mentre il `ds.b 40*256` è come un piccolo biglietto da 100.000 lire. Il risultato è lo stesso, ma il file è più snello.

Da notare che il `ds.b 40*256` non è seguito dal `,0` come nel `DCB`, infatti il `DS` indica sempre degli zeri, mentre il `DCB` può mettere in memoria un qualsiasi valore ripetuto X volte.

Ora abbiamo il "pezzo di carta" dove scrivere le nostre cose, ma non abbiamo né il font né la routine che stampa. Vediamo cosa è un *font* e come è fatto. Un font è un file che contiene le parole e i numeri necessari per scrivere, e può essere di vari formati. Il font non è altro che una fila di caratteri uno sotto l'altro, precisamente sono *tutti* i caratteri in fila: "ABCDEFGHI...". Certi font sono disegnati in .IFF, cioè una schermata con i caratteri:

```
 ------------
|ABCDEFGHIJKL|
|MNOPQRSTUVWX|
|YZ1234567890|
|            |
|            |
 ------------
```

Il disegno viene poi convertito in RAW, e i caratteri sono presi da quella figura e copiati nel bitplane: se va stampata una "A", viene copiata dal FONT in RAW al BITPLANE, con dei move, e la "A" appare sul bitplane. Così ogni volta che serve una "A" sappiamo dove si trova e la copiamo dal FONT, così per le altre lettere. Parliamo del sistema usato nei font 8x8 in questo corso: i caratteri occupano 8 pixel*8pixel, dunque sono grandi come il FONT del kickstart. In realtà sono più stretti in quanto devono contenere anche la "spaziatura" di un pixel tra una parola e l'altra, o la scrittura sembrerebbe in corsivo!! I caratteri poi sono messi nell'ordine "giusto", ossia che rispetta quello ASCII, che è il seguente:

```
        dc.b    $1f,' !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNò
        dc.b    'PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~',$7F
```

Il $1f iniziale e il $7f finale indicano che il primo carattere, lo SPAZIO, è quello dopo il $1f, ossia il $20, segue "!" che è il $21 eccetera, mentre dopo gli ultimi caratteri si arriva al $7f. Questo è per darvi un'idea della disposizione dei caratteri ASCII. Abbiamo già parlato del fatto che i numeri possono essere anche caratteri ASCII, basta provare con un *<?$21>*, verificando che il risultato viene dato in esadecimale ($), decimale, ASCII "...!", e binario. Abbiamo anche visto che un:

```
        dc.b    "CANE"
```

è equivalente ad un:

```
        dc.b    $63,$61,$6e,$65
```

Infatti "C" in memoria è $63, "A" è $61 eccetera. Ogni carattere infatti occupa un byte in memoria, e un testo lungo 5000 bytes contiene 5000 caratteri. Ritornando al nostro font, immaginatevi una figura larga solo 8 pixel, e alta abbastanza da contenere tutti i caratteri posti l'uno sotto l'altro:

```
!
"
#
$
%
&
'
(
)
*
+
,
-
.
/
0
1
2
3
4
5
6
7
8
9
:
;
<
=
>
?
@
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O

ECCETERA ECCETERA.....
```

Il font 8x8 che usiamo nel corso non è altro che una figura del genere in RAW. In realtà questo tipo di font viene fatto normalmente con un apposito EDITOR, un programma dedicato al disegno di questi font 8x8 ad un colore. Per i font più grandi e colorati però conviene disegnare le lettere in una figura, normalmente 320x256, e usare una routine propria per prelevare i caratteri da stampare. Per cominciare però vediamo il font più semplice come viene stampato a video: innanzitutto bisogna preparare una stringa di testo con le parole da stampare, ad esempio:

```
1       dc.b    "Prima scritta!"        ; nota: si possono usare '' oppure ""
2
3       EVEN                            ; ossia allinea ad indirizzo PARI
```

La direttiva EVEN serve ad evitare gli indirizzi dispari per le istruzioni o i dati che si trovano sotto il dc.b. Le stringhe di testo sono composte di bytes e può succedere che siano un numero dispari, in tal caso la label sottostante sarà ad un indirizzo dispari, e questo può generare errori di assemblaggio: infatti, nel 68000, le istruzioni devono sempre essere ad indirizzi pari, e anche i dati dovrebbero essere ad indirizzi pari per evitare GURU MEDITATION in fase di esecuzione, infatti un MOVE.L o MOVE.W eseguito su un indirizzo dispari causa un bel Crash con GURU MEDITATION ed esplosioni. Ricordatevi dunque di mettere sempre un EVEN al termine di una stringa di testo, o di accertarvi che sia pari. Potete anche aggiungere uno zero in più al termine della stringa per pareggiare il conto, come ho fatto per GfxName:

```
GfxName:
        dc.b    "graphics.library",0,0

Potete scrivere anche:

GfxName:
        dc.b    "graphics.library",0
        even
```

Infatti basta uno zero alla fine del testo, l'altro lo metterà EVEN. Dunque, una volta stabilita la stringa di testo da visualizzare, basta vedere come copiare i caratteri giusti al posto giusto. Vi propongo già la routine che stampa un carattere:

```
PRINT:
        LEA     TESTO(PC),A0    ; Indirizzo del testo da stampare in a0
        LEA     BITPLANE,A3     ; Indirizzo del bitplane destinazione in a3
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0),D2         ; Prossimo carattere in d2
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che è $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W  #8,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D2,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,40*3(A3)  ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,40*4(A3)  ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,40*5(A3)  ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,40*6(A3)  ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,40*7(A3)  ; stampa LA LINEA 8  " "

        RTS
```

Avete già capito??? Analizziamola punto per punto:

```
        LEA     TESTO(PC),A0    ; Indirizzo del testo da stampare in a0
        LEA     BITPLANE,A3     ; Indirizzo del bitplane destinazione in a3
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0),D2         ; Prossimo carattere in d2
```

Fino a qua non ci sono problemi, abbiamo in d2 il valore del carattere, se fosse una "A", allora abbiamo $41 in d2

```
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che è $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
```

Anche qua cosa succede è chiaro, vediamo perché sottraiamo 32 ($20):

```
        MULU.W  #8,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D2,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...
```

Questa operazione porta ad avere in A2 l'indirizzo del carattere "A" presente nel font, ossia l'indirizzo da dove dobbiamo "prendere" il carattere per copiarlo nel bitplane che stiamo visualizzando. Vediamo cosa è successo: ricordate che i caratteri sono stati messi nel font nello stesso ordine dello standard ASCII? Per cui, disponendo del valore ASCII del carattere, in questo caso $41 per la "A", possiamo individuare a che distanza dall'inizio del FONT si trova la "A" in RAW! Se ogni carattere è di 8x8 pixel, significa che è lungo 8 bit, ossia un byte a linea * 8 linee, in totale 8 bytes. Dunque lo spazio (il primo carattere nel FONT) si trova all'inizio del FONT stesso e finisce al byte 8, dove inizia "!" (il secondo) , e così via. Avendo sottratto $20 al valore ASCII, il valore dello spazio diventerà $00, il carattere successivo "!" $01, eccetera (la "A" risulterà $21), dunque basta moltiplicare per 8 il numero ottenuto dopo la sottrazione per ricavare la distanza, dall'inizio del FONT, del carattere in questione!!! Rivediamo il passaggio:

```
1       SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
2                               ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
3                               ; DELLO SPAZIO (che è $20), in $00, quello
4                               ; DELL'ASTERISCO ($21), in $01...
5       MULU.W  #8,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
6                               ; essendo i caratteri alti 8 pixel
```

Ora in D2 abbiamo la distanza (l'offset) dell'inizio del carattere dall'inizio del FONT! Ora per trovare l'indirizzo effettivo del carattere, aggiungiamo la "distanza dall'inizio" che abbiamo in D2 all'indirizzo del FONT:

```
1       MOVE.L  D2,A2
2       ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...
```

Ora abbiamo in a2 l'indirizzo dove si trova il nostro carattere da copiare, ad esempio la "A". Basterà ora copiarla da FONT allo schermo, cioè al BITPLANE 320x256, in cui ogni linea è lunga 40 bytes:

```
1                               ; STAMPIAMO IL CARATTERE LINEA PER LINEA
2       MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
3       MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2  " "
4       MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3  " "
5       MOVE.B  (A2)+,40*3(A3)  ; stampa LA LINEA 4  " "
6       MOVE.B  (A2)+,40*4(A3)  ; stampa LA LINEA 5  " "
7       MOVE.B  (A2)+,40*5(A3)  ; stampa LA LINEA 6  " "
8       MOVE.B  (A2)+,40*6(A3)  ; stampa LA LINEA 7  " "
9       MOVE.B  (A2)+,40*7(A3)  ; stampa LA LINEA 8  " "
```

La copia avviene per "linee", infatti il carattere è alto 8 linee, ognuna delle quali è larga 8 bit (1 byte):

```
12345678

...###.. linea  1 - 8 bit, 1 byte
..#...#. 2
..#...#. 3
..#####. 4
..#...#. 5
..#...#. 6
..#...#. 7
........ 8
```

Dunque per copiarlo una linea alla volta occorre copiarne un byte alla volta. Ma lo schermo destinazione è largo 40 bytes per linea, e dobbiamo considerare che ogni linea deve essere allineata l'una sotto l'altra, se non saltiamo 40 bytes ogni volta copieremmo così il carattere:

```
...###....#...#...#...#...#####...#...#...#...#...#...#.........
```

Invece dobbiamo copiare un byte, poi ANDARE A CAPO saltando 40 bytes, e copiare un altro byte:

```
1       MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
```

Sul monitor:

```
...###..
```

```
1       MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2 (40 bytes dopo)
```

Sul monitor:

```
...###..
..#...#.
```

```
1       MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3 (80 bytes dopo)
```

Sul monitor:

```
...###..
..#...#.
..#...#.
```

Eccetera. Per uno schermo largo 80 bytes (640x256 HIRES) basterebbe cambiare la routine così:

```
1       MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
2       MOVE.B  (A2)+,80(A3)    ; stampa LA LINEA 2  " "
3       MOVE.B  (A2)+,80*2(A3)  ; stampa LA LINEA 3  " "
4       MOVE.B  (A2)+,80*3(A3)  ; stampa LA LINEA 4  " "
5       MOVE.B  (A2)+,80*4(A3)  ; stampa LA LINEA 5  " "
6       MOVE.B  (A2)+,80*5(A3)  ; stampa LA LINEA 6  " "
7       MOVE.B  (A2)+,80*6(A3)  ; stampa LA LINEA 7  " "
8       MOVE.B  (A2)+,80*7(A3)  ; stampa LA LINEA 8  " "
```

Vediamo in pratica la stampa di questa "A" su un bitplane in `Lezione6a.s`.

Ora passeremo a stampare un'intera riga di testo con `Lezione6b.s`.

E infine stampiamo quante righe vogliamo in `Lezione6c.s`. Questa routine è quella *definitiva*, che potete usare quando volete scrivere qualcosa a video.

Perché non disegnarsi il proprio font di caratteri? In `Lezione6c2.s` il FONT è nel listato in `dc.b` come questo esempio:

```
1  ; "B"
2          dc.b    %01111110
3          dc.b    %01100011
4          dc.b    %01100011
5          dc.b    %01111110
6          dc.b    %01100011
7          dc.b    %01100011
8          dc.b    %01111110
9          dc.b    %00000000
```

I caratteri sono messi in memoria con dei `dc.b %` (binario). Potete cambiare ogni singolo carattere come volete. Se fate un vostro font, salvatelo su un disco formattato o sull'HARD DISK!

Ora abbiamo l'occasione di provare una cosa che non abbiamo mai fatto prima: nello stesso schermo proviamo a far convivere una figura in LOWRES a 8 colori e un bitplane in HIRES. L'Amiga infatti può visualizzare contemporaneamente diverse risoluzioni video, (cosa che non mi risulta possa fare il PC MSDOS), basta mettere un `WAIT` nella copperlist e ridefinire sotto di esso il **??**, proprio come se definissimo i colori per fare una sfumatura! Per esempio potremmo visualizzare dalla prima linea alla linea $50 una figura in HAM a 4096 colori in LOWRES, sotto di essa una in HIRES a 16 colori, sotto ancora una in LOWRES a 32 colori, e così via. In alcuni giochi per esempio la schermata dove si muovono i personaggi è in LOWRES, mentre il pannello

con il punteggio e simili è in HIRES (vedi AGONY). Visualizziamo subito la figura in LOWRES sopra una in HIRES in `Lezione6d.s`.

Vediamo ora un "trucchetto" che ci permette di ottenere un effetto di "rilievo" alle parole che stampiamo: in `Lezione6e.s` attiviamo 2 bitplane anziché 1 e sovrapponiamo il secondo al primo, ma il secondo spostato in basso di una linea. Cosa succede se mettiamo due immagini uguali trasparenti l'una sull'altra? L'immagine si sdoppia!!! E se scegliamo i colori giusti, facendo più chiaro lo sdoppiamento in "alto" e più scuro quello in "basso" cosa succede? Che abbiamo capito come funziona `Lezione6e.s`.

A proposito di sovrapposizioni, perché non attivare un bitplane "sopra" una figura per scriverci?? Vediamo in `Lezione6f.s` cosa succede.

In `Lezione6g.s` viene evidenziato l'effetto *trasparenza* muovendo la scritta sopra la figura.

In `Lezione6h.s`, invece, troverete un modo per stampare testi a 3 colori, sovrapponendo due testi in due bitplanes.

In `Lezione6i.s` viene fatto lampeggiare uno dei 3 colori del testo, usando una *tabella* di valori predefiniti. Abbiamo già parlato di tabelle nella LEZIONE1, ora vediamo in pratica il vantaggio che portano.

In `Lezione6l.s` viene usata una variazione della routine che legge da una tabella per variare un colore; la variazione consiste nel fatto che anziché leggere dall'inizio alla fine della tabella e ripartire da capo, rilegge la tabella all'indietro, cioè dalla fine all'inizio.

Le tabelle possono essere utili o indispensabili per molti usi, ad esempio per simulare movimenti di rimbalzi o di oscillazioni. Vediamo in pratica la superiorità dell'uso di una tabella rispetto a semplici `ADD` e `SUB` nel movimento di una figura in `Lezione6m.s`.

A proposito di movimento, per ora abbiamo visto lo scroll orizzontale tramite il `BPLCON1` (`$dff102`) che permette uno scorrimento massimo di 16 pixel. Ma allora come si fa a scorrere lo schermo a destra e a sinistra quanto vogliamo?? La risposta è abbastanza semplice: basta usare anche i puntatori ai bitplanes! Infatti, tramite i puntatori ai bitplanes abbiamo già visto che possiamo scorrere in alto e in basso, basta aggiungere o sottrarre la lunghezza di una linea (40 in lowres e 80 in HIRES). Ma possiamo scorrere anche in avanti e indietro, per la precisione a "scatti" di 8 pixel alla volta, basta sottrarre o aggiungere 1 al puntatore bitplane e abbiamo spostato a destra o a sinistra la figura di un byte, ossia 8 bit, ossia 8 pixel. Se possiamo scorrere di 8 pixel alla volta con i Bitplane Pointers e di 1 alla volta con il `$dff102` (`BPLCON1`), basterà scorrere 8 pixel uno alla volta col `$dff102`, appunto, poi "scattare" 8 pixel più avanti con un:

```
1   subq.l #1,BITPLANEPOINTER
```

E azzerare contemporaneamente il `BPLCON1` (`$dff102`), andando al nono pixel, dopodiché scorrere di altri 8 pixel col `$dff102` di un pixel alla volta, giungendo al pixel 9+8= 11, poi scattare in avanti di 8 pixel col Bitplane Pointer eccetera. Negli esempi però, considerando che il `$dff102` può scorrere fino ad un massimo di $FF, ossia da 0 a 15, e non solo da 0 a 7, ho adottato questa tecnica: per scorrere di 16 pixel alla volta basta aggiungere o sottrarre 2 ai puntatori bitplane (dato che con 1 spostavamo la PIC di 8 pixel) Dunque scorro un pixel alla volta col `$dff102` usando la sua possibilità massima, cioè da $00 a $FF, totale 16 posizioni, dopodiché "scatto" ai 16 pixel seguenti con un `ADDQ` o `SUBQ #2,BITPLANEPOINTERS`. Ecco una routine che scorre verso destra un bitplane di un pixel alla volta per quanti pixel vogliamo: considerate che `MIOBPCON1` è il byte del `$dff102`.

```
1   Destra:
2           CMP.B   #$ff,MIOBPCON1  ; siamo arrivati al massimo scorrimento? (15)
3           BNE.s   CON1ADDA        ; se non ancora, scorri in avanti di 1
4                                   ; con il BPLCON1
5
6   ;       Legge l'indirizzo del bitplane
7
8           LEA     BPLPOINTERS,A1  ; Con queste 4 istruzioni preleviamo dalla
```

```
9           move.w  2(a1),d0        ; copperlist l'indirizzo dove sta puntando
10          swap    d0              ; attualmente il $dff0e0 e lo poiniamo in d0
11          move.w  6(a1),d0
12
13  ;       Scorre a destra di 16 pixel col puntatore bitplane
14
15          subq.l  #2,d0           ; punta 16 bit più indietro ( la PIC scorre
16                                  ; verso destra di 16 pixel)
17
18  ;       Fa ripartire da zero il BPLCON1
19
20          clr.b   MIOBPCON1       ; azzera lo scroll hardware BPLCON1 ($dff102)
21                                  ; infatti abbiamo "saltato" 16 pixel con il
22                                  ; bitplane pointer, ora dobbiamo ricominciare
23                                  ; da zero con il $dff102 per scattare a
24                                  ; destra di un pixel alla volta.
25
26          move.w  d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
27          swap    d0              ; scambia le 2 word
28          move.w  d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
29          rts                     ; esci dalla routine
30
31  CON1ADDA:
32          add.b   #$11,MIOBPCON1  ; scorri a destra di 1 pixel la figura
33          rts                     ; esci dalla routine
```

La routine aumenta di uno il BPLCON1 ($dff102), facendolo passare per le 16 posizioni possibili: 00, 11, 22, 33, 44, 55, 66, 77, 88, 99, aa, bb, cc, dd, ee, ff dopodiché salta al pixel ff+1 facendo 2 operazioni:

1. Puntare 2 bytes (1 word, 16 bits) più indietro i puntatori bitplanes, facendo scorrere a destra di 16 pixel la figura (dunque 1 pixel dopo la posizione `$FF`, ossia 15 raggiunta il fotogramma precedente dal `$dff102`
2. Azzerare il `$dff102`, dato che abbiamo "saltato" 16 pixel, altrimenti si sommerebbero i 16 pixel aggiunti col Puntatore Bitplane e i 15 ($FF) raggiunti col `$dff102` (`BPLCON1`). Invece azzerando il `BPLCON1` ripartiamo da $00+16= sedicesimo pixel, dopodiché andremo ai seguenti 15 con il `BPLCON1`, lasciando inalterato il puntatore bitplane.

Se non fosse ancora chiaro, seguite questo schemino, tenendo presente che `#` è la "figura" che spostiamo verso destra:

```
1                                   ; VAL. BPLCON1  - BYTE SOTTRATTI AI PUNT. PLANE
2
3   #                               ;       $00     -       0       - tot. pixel:
4    #                              ;       $11     -       0       -       1
5     #                             ;       $22     -       0       -       2
6      #                            ;       $33     -       0       -       3
7       #                           ;       $44     -       0       -       4
8        #                          ;       $55     -       0       -       5
9         #                         ;       $66     -       0       -       6
10         #                        ;       $77     -       0       -       7
11          #                       ;       $88     -       0       -       8
12           #                      ;       $99     -       0       -       9
13            #                     ;       $aa     -       0       -       10
14             #                    ;       $bb     -       0       -       11
15              #                   ;       $cc     -       0       -       12
16               #                  ;       $dd     -       0       -       13
17                #                 ;       $ee     -       0       -       14
18                 #                ;       $ff     -       0       -       15
19                  #               ;       $00     -       2       -       16
20                   #              ;       $11     -       2       -       17
21                    #             ;       $22     -       2       -       18
22                     #            ;       $33     -       2       -       19
23                      #           ;       $44     -       2       -       20
24                       #          ;       $55     -       2       -       21
25                        #         ;       $66     -       2       -       22
26                         #        ;       $77     -       2       -       23
```

```
27
28 eccetera....
```

Questo schema parla da solo: per esempio se vogliamo scorrere verso destra di 22 pixel un bitplane basta sottrarre 2 al bitplane pointer e mettere $66 al BPLCON1 ($dff102).

Per scorrere a sinistra dovremo invece aggiungere 2 ai puntatori bitplanes ogni 16 pixel e procedere al contrario con il $dff102: $ff,$ee,$dd...

Vediamo in Lezione6n.s la routine in funzione. Noterete un imprevisto: sul lato sinistro avviene un disturbo a scatti; questo non è dovuto ad errori nella routine, ma ad una caratteristica dell'hardware di Amiga, per toglierlo basta un piccolo accorgimento già presente nelle modifiche consigliate del listato stesso.

Già che sappiamo scorrere quanto vogliamo anche in orizzontale, perché non scorrere un bitplane più grande della finestra video?? Esattamente facciamo scorrere uno schermo largo 640 pixel in uno largo 320 spostandolo a destra e sinistra, tutto questo in Lezione6o.s.

Abbiamo già visto per le tabelle l'utilizzo di una longword come puntatore ad un indirizzo:

```
1 PUNTATORE:
2         DC.L    TABELLA
```

Nella longword "puntatore" viene assemblato l'indirizzo di tabella, per cui possiamo tenere "il conto" di dove siamo arrivati nella tabella aggiungendo o sottraendo la lunghezza di un elemento della tabella. Dobbiamo salvare l'indirizzo a cui siamo arrivati ogni volta perché la routine viene eseguita ogni fotogramma e non continuamente, dunque possono essere eseguite anche altre routine prima che quella routine sia eseguita nuovamente. Quando questa routine viene rieseguita, deve continuare a prelevare valori dalla tabella da dove era rimasta la volta prima, e lo può fare leggendo l'indirizzo in PUNTATORE con un semplice:

```
1         MOVE.L  PUNTATORE(PC),d0        ; In d0 l'indirizzo dove siamo arrivati
2                                         ; l'ultima volta.
```

Prima di uscire dalla routine basterà salvare l'ultima posizione. Questo espediente può essere usato per molti scopi, per esempio per poter stampare un carattere solo ogni fotogramma, anziché stampare tutto il testo e poi vederlo. Per fare ciò basta modificare la routine PRINT: e farsi due puntatori: uno che punti all'ultimo carattere stampato, ed uno che punti all'ultimo indirizzo nel bitplane dove abbiamo stampato l'ultimo carattere. In questo modo è come se stampassimo un carattere, congelassimo la routine per tutto un fotogramma, la riattivassimo per stampare un carattere, poi la ricongelassimo eccetera. In realtà anziché congelarla la eseguiamo per stampare un solo carattere, poi salviamo il punto dove siamo arrivati, usciamo dalla routine, aspettiamo che passi il fotogramma, rieseguiamo la routine ripartendo dal punto dove siamo arrivati, risalviamo tutto, usciamo eccetera. Il listato che mette in pratica questa possibilità è Lezione6p.s.

In un bitplane oltre che stampare testo possiamo anche creare disegni con routine apposite, come scacchiere, trame e tessiture. Basta porre ad 1 i bit giusti!!! In Lezione6q.s ci sono delle routine esempio.

Siamo giunti alla fine della LEZIONE6, non ci resta che mettere insieme i listati e le "novità" di questa lezione nel consueto listatone finale di esempio con musica: Lezione6r.s.

Ora passeremo allo studio degli sprite. Quello che dovete fare è caricare la LEZIONE7.TXT, dopodiché dovete cambiare il path per caricare gli incbin dei suoi listati, con *<V DF0:SORGENTI3>*. I sorgenti, infatti, si trovano nella directory SORGENTI3 del disco 1.

CAPITOLO 7

LEZIONE 7 - GLI SPRITES E I DISPOSITIVI DI INPUT

In questa lezione parleremo degli sprites, del joystick e delle istruzioni 68000 riguardanti le operazioni sui bit come `AND`, `OR`, `EOR`, `NOT`, `LSR`, `ROL`...

Ricordatevi di scrivere `V df0:SORGENTI3` per poter caricare i file `.raw` dalla directory dove si trovano i listati di questa lezione.

Gli sprite sono oggetti grafici di una dimensione precisa, larghi al massimo 16 pixel, che si possono muovere per lo schermo indipendentemente dai bitplanes, per esempio il puntatore a freccia che muovete col mouse per selezionare dai menu o premere "pulsanti" è uno sprite gestito dal sistema operativo, che può muoversi dove vuole senza curarsi dei bitplanes che gli stanno "sotto".

Gli sprite si potrebbero considerare come immagini "fantasma" che si aggirano "sopra" i bitplanes, ma non tutte le cose che si muovono sono sprites! Infatti ci possono essere solo 8 sprites al massimo, essendoci sono solo 8 puntatori in copperlist per gli sprites:

```
1   COPPERLIST:
2
3   SpritePointers:
4           dc.w    $120,0,$122,0   ; Puntatore per lo Sprite 0
5           dc.w    $124,0,$126,0   ; Puntatore per lo Sprite 1
6           dc.w    $128,0,$12a,0   ; ""    ""      ""      2
7           dc.w    $12c,0,$12e,0   ; ""    ""      ""      3
8           dc.w    $130,0,$132,0   ; ""    ""      ""      4
9           dc.w    $134,0,$136,0   ; ""    ""      ""      5
10          dc.w    $138,0,$13a,0   ; ""    ""      ""      6
11          dc.w    $13c,0,$13e,0   ; ""    ""      ""      7
```

I puntatori agli sprite si chiamano registri `SPRxPT` (al posto della `x` si mette il numero dello sprite: abbiamo dunque `SPR0PT`, `SPR1PT`, ... `SPR7PT` e quando parliamo di `SPRxPT` ci riferiamo in generale a tutti gli 8 puntatori).

Per ora li abbiamo messi nella copperlist azzerati solo per evitare che questi oggetti "fantasma" saltellino sulle nostre figure senza controllo. Gli sprite sono isolati dal resto dello schermo, come fossero in una "busta trasparente" applicata sopra il monitor, infatti la risoluzione degli sprite è sempre il lowres, 320x256, anche se i bitplanes sottostanti sono in hires o interlacciati.

Una verifica che gli sprite non fanno parte dei bitplane è che per muoverli non occorre cancellarli e ridisegnarli più avanti ogni volta, come avremmo invece dovuto fare per spostare un pezzo di grafica in un bitplane.

Per muovere uno sprite basta cambiare le sue coordinate agendo con poche e veloci istruzioni su appositi byte dedicati a questo compito che si trovano all'inizio della struttura dati dello sprite stesso.

Quando gli sprites non bastano a fare astronavi e ometti in un gioco viene usato il blitter per copiare blocchi di grafica (*bob*), che vedremo in seguito. Come già detto, la dimensione di uno sprite è di 16 pixel di larghezza, mentre l'altezza può essere scelta a piacere, anche tutto lo schermo, cioè 256 linee. Per fare un mostro di fine livello si potrebbero usare tutti e 8 gli sprites affiancati, raggiungendo la larghezza totale di 16*8=128 pixel.

Il problema è che tale mostro sarebbe poco colorato per i tempi che corrono, infatti uno sprite può avere 3 colori al massimo, dato che il "quarto" è la parte "trasparente", ossia la parte in cui traspare lo sfondo, ossia i bitplanes.

La caratteristica degli sprites è che sono semplici da fare e da animare. Infatti lo sprite si può disegnare con un programma da disegno, basta che sia largo non più di 16 pixel e che abbia 3 colori più lo sfondo, ossia 4, e può essere convertito in *SPRITE* dall'*IFFCONVERTER KEFCON*.

Oppure si può disegnare direttamente in binario, come abbiamo visto per il font 8x8:

```
1                 — piano 1 —       — piano 2 —     ; la sovrapposizione di
2                                                   ; questi 2 "piani" di
3       dc.w    %0111110000000000,%0111110000000000 ; bit determina il
4       dc.w    %1000001000000000,%1111111000000000 ; colore.
5       dc.w    %1111010000000000,%1000110000000000 ; Questa è la freccia
6       dc.w    %1111101000000000,%1000011000000000 ; puntatore di default
7       dc.w    %1111110100000000,%1001001100000000 ; del kickstart 1.3, la
8       dc.w    %1110111010000000,%1010100110000000 ; riconoscete??
9       dc.w    %0100011101000000,%0100010011000000
10      dc.w    %0000001110100000,%0000001001100000
11      dc.w    %0000000111100000,%0000000100100000
12      dc.w    %0000000011000000,%0000000011000000
13      dc.w    %0000000000000000,%0000000000000000
14
15      dc.w    0,0 ; Due word azzerate indicano la fine dello sprite.
```

In questo caso la larghezza è di 16 pixel e non di 8 come nel font 8x8, percui lo disegnamo in una word (`dc.w`) e non in un byte. Inoltre ha 3 colori più il trasparente, ossia 4 possibilità come una figura a 2 bitplanes, dunque servono un paio di "piani" proprio come per i bitplanes, e la loro sovrapposizione determinerà il colore, che può essere:

```
          Piano 1 - Piano 2
binario:     0    -    0    = COLORE 0 (TRASPARENTE)
binario:     1    -    0    = COLORE 1
binario:     0    -    1    = COLORE 2
binario:     1    -    1    = COLORE 3
```

Infatti, come abbiamo già visto, con 2 piani di bit si possono formare 4 combinazioni diverse: %00,%01,%10,%11

Per decidere la posizione dello sprite basta inserire le coordinate X ed Y nei primi byte dello sprite stesso. Infatti, prima dei dati del disegno, lo sprite è composto da 4 byte, ossia 2 word, dette *word di controllo*, ed in questi byte vanno scritte le coordinate sullo schermo dello sprite. Per essere più esatti, il primo byte, detto `VSTART`, contiene la posizione verticale di inizio dello sprite; il secondo byte invece contiene la posizione orizzontale (`HSTART`). Il terzo contiene la posizione della fine dello sprite in senso verticale: per determinarla basta aggiungere l'altezza dello sprite alla posizione inizio, e come risultato avremo la posizione verticale dove finisce lo sprite.

Il quarto byte contiene dei bit per funzioni speciali che vederemo. VSTART e HSTART (Vertical Start e Horizontal Start) dunque sono le coordinate dell'angolo in alto a sinistra dove inizia lo sprite:

```
#....
.....
.....
.....
.....
```

Mentre VSTOP è la posizione verticale dove termina lo sprite:

```
.....
.....
.....
.....
#####    -> linea verticale indicata da VSTOP.
```

Per esempio, uno sprite visualizzato alla posizione XX=$90 e YY=$50, lungo 20 pixel, comincerebbe così:

```
1              ;IYIX  FY      - IY=Inizio Y, IX=Inizio X, FY=Fine Y
2  SPRITE:
3      dc.w    $5090,$6400     ;Y=$50, X=$90, altezza= $50+20, cioè $64
4  ; da qua iniziano i dati dei 2 piani dello sprite
5      dc.w    %0000000000000000,%0000110000110000
6      dc.w    %0000000000000000,%0000011001100000
7      ...
8      dc.w    0,0     ; fine dello sprite
```

Infatti il primo byte, VSTART, è a $50, il secondo, HSTART, è a $90, mentre il terzo, la posizione verticale di fine sprite, è a $64, ossia a $50+20, la posizione inizio più la lunghezza dello sprite. Il quarto byte per ora lo lasciamo a zero, vedremo in seguito a cosa serve. Posso premettere che il byte HSTART, ossia quello che si occupa della posizione orizzontale, fa spostare lo sprite a "scatti" di 2 pixel alla volta, per cui muovendo uno sprite dalla posizione $50 alla posizione $51, ad esempio, scatterebbe a destra di 2 pixel, e non di uno: vedremo che usando un bit del quarto byte si può far scorrere lo sprite di un pixel alla volta orizzontalmente.

Per quanto riguarda la posizione verticale, invece, lo scorrimento avviene già con VSTART/-VSTOP a scatti di un pixel, ma la limitazione è la linea video $FF, oltre la quale si può andare usando un altro dei bit del quarto byte. Per ragioni di semplicità nei primi esempi sposteremo gli sprite solamente agendo sui byte HSTART, VSTART e VSTOP, ossia con le limitazioni di uno scorrimento orizzontale a "scatti" di due pixel alla volta. Solo in un secondo momento vedremo come fare scorrimenti più fluidi. Ricordatevi dunque della particolarità che, ad esempio, con un:

```
1      ADDQ.B #1,HSTART
```

spostiamo lo sprite di 2 pixel e non di uno. Per agire sui 3 byte VSTART, HSTART, VSTOP si potrebbe fare così:

```
1      MOVE.B #$50,SPRITE      ; VSTART = $50
2      MOVE.B #$90,SPRITE+1    ; HSTART = $90
3      MOVE.B #$64,SPRITE+2    ; VSTOP  = $64 ($50+20)
```

Oppure si può definire una label per ogni byte per renderlo più chiaro:

```
1  SPRITE:
2  VSTART:         ; posizione inizio VERTICALE
3      dc.b $50
4  HSTART:         ; posizione inizio ORIZZONTALE
5      dc.b $90
6  VSTOP:
7      dc.b $64    ; posizione fine VERTICALE
```

```
8       dc.b $00    ; byte per funzioni speciali azzerato
9
10  ; da qua iniziano i dati dei 2 piani dello sprite
11
12      dc.w        %0000000000000000,%0000110000110000
13      dc.w        %0000000000000000,%0000011001100000
14      ...
15      dc.w        0,0 ; fine dello sprite
```

In questo caso agiremmo sulle label `VSTART`, `HSTART` e `VSTOP`:

```
1       ADDQ.B #1,HSTART    ; sposta lo sprite a destra di 2 pixel
2                           ; (2 pixel e non 1 per le ragioni descritte)
3       SUBQ.B #1,HSTART    ; sposta lo sprite a sinistra di 2 pixel
```

Per spostare in basso o in alto lo sprite dovremmo però ricordarci di modificare sia `VSTART` che `VSTOP`, perché è ovvio che se spostiamo in basso o in alto lo sprite si sposta sia il primo pixel a sinistra che l'ultimo:

```
1       ADDQ.B #1, VSTART   ; \ sposta lo sprite in basso di 1 pixel
2       ADDQ.B #1, VSTOP    ; /
3
4       SUBQ.B #1, VSTART   ; \ sposta lo sprite in alto di 1 pixel
5       SUBQ.B #1, VSTOP    ; /
```

Ricapitolando questa è la struttura dello sprite:

```
prima word di controllo,         seconda word di controllo
prima   linea (.w) del piano 1,  prima   linea (.w) del piano 2
seconda linea (.w) del piano 1,  seconda linea (.w) del piano 2
terza   linea (.w) del piano 1,  terza   linea (.w) del piano 2
quarta  linea (.w) del piano 1,  quarta  linea (.w) del piano 2
quinta  linea (.w) del piano 1,  quinta  linea (.w) del piano 2
...
dc.w    0,0  ; l'ultima riga deve contenere due zeri
```

I dati dello sprite sono divisi in piano 1 e piano 2 solo per indicare che la loro sovrapposizione determina i 3 colori più il trasparente in maniera analoga ai bitplanes dello schermo, ma non vanno confusi con questi ultimi!

7.1 I colori degli sprite

Per definire i colori degli sprite bisogna usare gli stessi registri colore usati dai bitplanes, in quanto l'Amiga ha solo 32 registri colore. I progettisti hanno pensato di far assumere agli sprites i colori dal 16 al 31, per cui se le figure non sono a 32 colori, ossia a 5 bitplanes, gli sprites possono avere colori diversi dalle figure. Altrimenti gli sprites avranno 16 colori in comune con la figura a 32 colori sottostante. Per ora vediamo come definire i colori del primo sprite (gli sprite sono numerati dallo 0 al 7):

```
COLORE 0 dello sprite 0 = TRASPARENZA, non va definito
COLORE 1 dello sprite 0 = COLOR17 ($dff1a2)
COLORE 2 dello sprite 0 = COLOR18 ($dff1a4)
COLORE 3 dello sprite 0 = COLOR19 ($dff1a6)
```

il colore 0, ossia il quarto, è la trasparenza e non occorre definirlo.

Vediamo, prima di procedere, il primo esempio di visualizzazione di uno sprite in `Lezione7a.s`. In questo esempio viene puntato il primo sprite, lasciando azzerati gli altri 7. Per puntare uno sprite bisogna fare come per i bitplanes, in quanto lo sprite ha i puntatori che funzionano allo stesso modo:

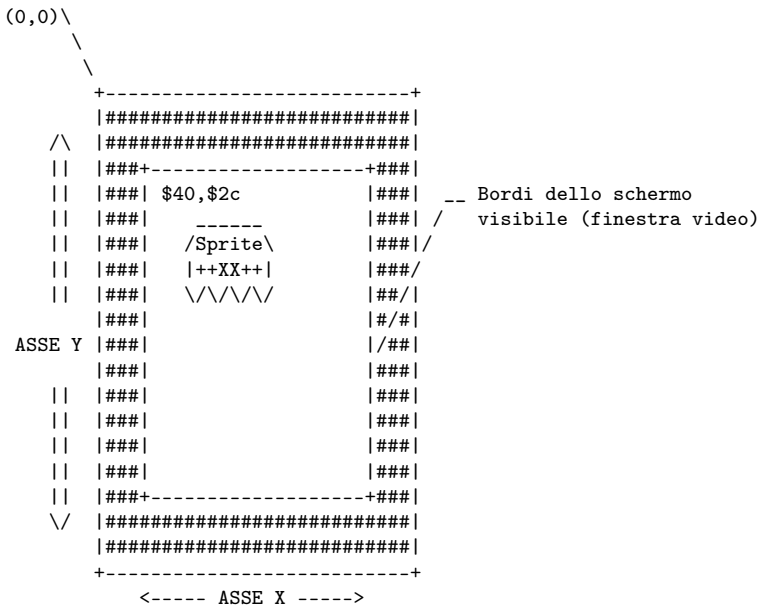
```
1     move.l #MIOSPRITE,d0     ; indirizzo dello sprite in d0
2     lea SpritePointers,a1    ; Puntatori in copperlist
3     move.w d0,6(a1)
4     swap d0
5     move.w d0,2(a1)
```

Va ricordato che per visualizzare gli sprite occorre aver "acceso" almeno un bitplane, con i bitplane disabilitati vengono disabilitati anche gli sprite. Allo stesso modo, uno sprite viene "tagliato" se va oltre la finestra video, definita col `DIWSTART` e `DIWSTOP`, essendo visualizzabile solo al suo interno. Da notare che per posizionare nello schermo 320x256 lo sprite, per esempio alla coordinata centrale (160,128) bisogna tener conto che la prima coordinata in alto a sinistra, dove inizia la finestra video, non è (0,0), ma $40,$2c per cui bisogna sommare $40 alla coordinata X e $2c alla coordinata Y. Infatti $40+160, $2c+128, corrispondono alla coordinata (160,128) di uno schermo 320x256 non overscan.

Non avendo ancora il controllo della posizione orizzontale a livello di 1 pixel, ma ogni 2 pixel, dobbiamo sommare non 160, ma 160/2, per individuare il centro dello schermo:

```
1  HSTART:
2     dc.b $40+(160/2)     ; posizionato al centro dello schermo
3     ...
```

Ecco uno schema dello schermo, in cui la parte visibile, ossia la finestra video, è bianca, mentre l'intero schermo, fuori dai bordi, che inizia con le coordinate 0,0 è fatto di ####. Si noti che la finestra video comincia dalle coordinate $40 XX e $2c YY.



La posizione *orizzontale* dello sprite può andare da 0 a 447, ma è chiaro che per essere visibile su schermo largo 320 pixel deve andare da 64 a 383. La posizione *verticale* dello sprite invece può andare da 0 a 262, ma per essere visibile su schermo largo PAL (256 linee) deve andare da 44 ($2c) alla fine dello schermo, 44+256= 300 ($12c). Per ora abbiamo raggiunto solo la posizione $FF, vedremo più avanti come andare fino alla $12c.

In `Lezione7b.s` lo sprite viene fatto scorrere sullo schermo con degli `ADD` e `SUB` sulle due word di controllo.

In `Lezione7c.s`, lo sprite viene spostato in orizzontale sullo schermo con delle tabelle di valori predefiniti anziché con `ADD` e `SUB`. In `Lezione7d.s` viene fatto saltellare in verticale. In

`Lezione7e.s` le due coordinate XX ed YY vengono definite da due tabelle per creare movimenti circolari, ad ellisse eccetera. In questo esempio viene anche spiegato come crearsi proprie tabelle!

Prima di procedere nella lettura caricate ed eseguite in altri buffer di testo questi esempi, leggendone i commenti finali.

Per ora abbiamo visualizzato un solo sprite, vediamo cosa occorre sapere se si visualizzano tutti e 8 gli sprite. Innanzitutto ogni sprite ha posizione indipendente rispetto agli altri, ed ha un `VSTART`, `HSTART` e `VSTOP` proprio nelle prime 2 word. Per quanto riguarda invece i colori (e anche altre proprietà degli sprite che vedremo successivamente, come per. es. le collisioni) gli sprite non sono totalmente indipendenti ma sono accoppiati a due a due. Ci sono dunque 4 coppie di due sprite: Sprite0+Sprite1, Sprite2+Sprite3, Sprite4+Sprite5, ed infine Sprite6+Sprite7. In tutto il resto della lezione, quando parleremo di *coppia di sprite* non intenderemo 2 sprite qualunque, ma una di queste 4 coppie.

Per i colori, bisogna tenere conto del fatto che gli sprite di una coppia hanno i colori in comune, ossia ogni coppia di sprite ha la sua palette (tavolozza) diversa da quella delle altre coppie. Sappiamo che i 3 colori dello sprite 0 sono definibili coi registri `COLOR17`, `COLOR18` e `COLOR19`. Questi 3 colori valgono anche per lo sprite "fratello", ossia lo sprite 1. Ogni coppia ha una palette colori diversa perché sono disponibili i registri colore dal 16 al 31, ossia 16 registri. Considerando che ogni sprite ha 4 colori (di cui 1 trasparente), servirebbero 8*4=32 registri, quando ne sono rimasti solo 16. Dunque, avendo 8 sprites con 4 colori ciascuno ecco da quali registi le coppie di sprite prendono i colori:

```
            Sprite  Valore binario  Registro di colore:
            ------  --------------  ------------------
Coppia 1:   0 o 1         00        Non Usato perché trasparente
                          01        Color17 - $dff1a2
                          10        Color18 - $dff1a4
                          11        Color19 - $dff1a6
Coppia 2:   2 o 3         00        Non Usato perché trasparente
                          01        Color21 - $dff1aa
                          10        Color22 - $dff1ac
                          11        Color23 - $dff1ae
Coppia 3:   4 o 5         00        Non Usato perché trasparente
                          01        Color25 - $dff1b2
                          10        Color26 - $dff1b4
                          11        Color27 - $dff1b6
Coppia 4:   6 o 7         00        Non Usato perché trasparente
                          01        Color29 - $dff1ba
                          10        Color30 - $dff1bc
                          11        Color31 - $dff1be
```

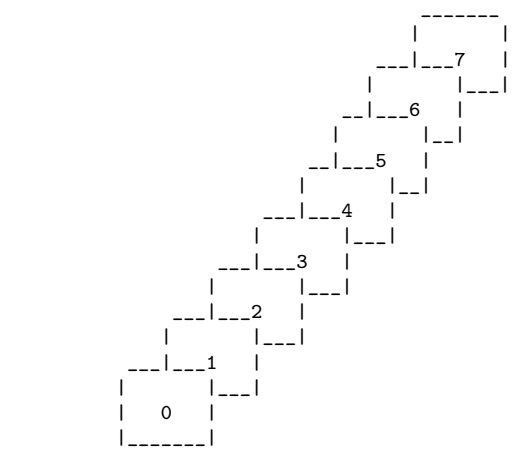
Facciamo un esempio pratico: nella copperlist per definire il colore degli 8 sprite è necessario fare questo:

```
1     dc.w $1A2,$F00  ; color17, — COLOR1 degli sprite0/1 —ROSSO
2     dc.w $1A4,$0F0  ; color18, — COLOR2 degli sprite0/1 —VERDE
3     dc.w $1A6,$FF0  ; color19, — COLOR3 degli sprite0/1 —GIALLO
4
5     dc.w $1AA,$FFF  ; color21, — COLOR1 degli sprite2/3 —BIANCO
6     dc.w $1AC,$0BD  ; color22, — COLOR2 degli sprite2/3 —ACQUA
7     dc.w $1AE,$D50  ; color23, — COLOR3 degli sprite2/3 —ARANCIO
8
9     dc.w $1B2,$00F  ; color25, — COLOR1 degli sprite4/5 —BLU
10    dc.w $1B4,$F0F  ; color26, — COLOR2 degli sprite4/5 —VIOLA
11    dc.w $1B6,$BBB  ; color27, — COLOR3 degli sprite4/5 —GRIGIO
12
13    dc.w $1BA,$8E0  ; color29, — COLOR1 degli sprite6/7 —VERDE CH.
14    dc.w $1BC,$a70  ; color30, — COLOR2 degli sprite6/7 —MARRONE
15    dc.w $1BE,$d00  ; color31, — COLOR3 degli sprite6/7 —ROSSO SC.
```

NOTA: Se impostate una figura a 2,4,8 o 16 colori come sottofondo, non ci sono problemi per la palette, ma se decidete di attivare uno schermo a 32 colori, ossia 5 bitplanes, la figura avrà in comune gli ultimi colori con gli sprite, per cui dovete fare in modo che i colori siano giusti sia per la figura che per lo sprite, che insomma il colore sia "multiuso".

7.2 La priorità video tra gli sprite

Quando ci sono due o più sprite sullo schermo può avvenire che degli sprite si sovrappongano. In questo caso viene coperto lo sprite con la priorità minore. La priorità tra gli sprite è sempre uguale, lo sprite con numero minore ha sempre priorità su quelli con numero maggiore, i quali rimangono "dietro". Di conseguenza lo sprite 0 può coprire tutti gli altri sprite, mentre lo sprite 7 può essere coperto da tutti gli altri. Ecco uno schemino:



Verifichiamo caricando ed eseguendo in un altro buffer di testo `Lezione7f.s`, il quale visualizza 8 sprite, e dopo la pressione del tasto sinistro del mouse li sovrappone per evidenziare le priorità. Tasto destro del mouse per uscire.

7.3 Sprite "attached"

Esiste anche una modalità di accoppiamento degli sprite a 2 a 2, l'uno sull'altro, che riduce il numero degli sprite disponibili alla metà, ossia a quattro, ma con 16 colori ciascuno anziché 4.(15 colori più il trasparente) Possono essere combinati solo in questo modo:

```
SPRITE0+SPRITE1      - Sprite ATTACCHED (attaccato) Numero 1
SPRITE2+SPRITE3      - Sprite ATTACCHED (attaccato) Numero 2
SPRITE4+SPRITE5      - Sprite ATTACCHED (attaccato) Numero 3
SPRITE6+SPRITE7      - Sprite ATTACCHED (attaccato) Numero 4
```

In pratica si attaccano gli sprite che in modalità normale fanno già coppia perché hanno la stessa palette. I 4 sprite "attaccati" condividono la stessa palette di 16 colori, dato che sono disponibili solo i registri colore dal Color16 al Color31. Gli sprite *attached* funzionano in questo modo: normalmente uno sprite ha al massimo 4 possibilità di sovrapposizione per i suoi piccoli "bitplanes", ossia %00 per il trasparente e %01,%10,%11 per gli altri 3 colori. La modalità *attached* fa sovrapporre i piani di bit dei due sprites per formare 16 possibilità, infatti ponendo i due piani del primo sopra i 2 piani del secondo si possono ottenere %1111 possibilità anziché

%11, ossia 16 anziché 4. Nella tabella che segue, nella colonna “valore binario”, sono elencate le varie possibilità di sovrapposizione e il relativo colore che ne risulta.

```
Colore    Valore   Numero del
Sprite    binario  registro colore
-------   ------  --------------
  0        0000    Color16 - NON USATO, è IL TRASPARENTE
  1        0001    Color17 - $dff1a2
  2        0010    Color18 - $dff1a4
  3        0011    Color19 - $dff1a6
  4        0100    Color20 - $dff1a8
  5        0101    Color21 - $dff1aa
  6        0110    Color22 - $dff1ac
  7        0111    Color23 - $dff1ae
  8        1000    Color24 - $dff1b0
  9        1001    Color25 - $dff1b2
  10       1010    Color26 - $dff1b4
  11       1011    Color27 - $dff1b6
  12       1100    Color28 - $dff1b8
  13       1101    Color29 - $dff1ba
  14       1110    Color30 - $dff1bc
  15       1111    Color31 - $dff1be
```

Dunque in COPPERLIST bisogna definirli in questo modo:

```
1     dc.w $1A2,$F00  ; color17, COLORE 1 per gli sprite attaccati
2     dc.w $1A4,$0F0  ; color18, COLORE 2 per gli sprite attaccati
3     dc.w $1A6,$FF0  ; color19, COLORE 3 per gli sprite attaccati
4     dc.w $1A8,$FF0  ; color20, COLORE 4 per gli sprite attaccati
5     dc.w $1AA,$FFF  ; color21, COLORE 5 per gli sprite attaccati
6     dc.w $1AC,$0BD  ; color22, COLORE 6 per gli sprite attaccati
7     dc.w $1AE,$D50  ; color23, COLORE 7 per gli sprite attaccati
8     dc.w $1B0,$D50  ; color24, COLORE 7 per gli sprite attaccati
9     dc.w $1B2,$00F  ; color25, COLORE 9 per gli sprite attaccati
10    dc.w $1B4,$F0F  ; color26, COLORE 10 per gli sprite attaccati
11    dc.w $1B6,$BBB  ; color27, COLORE 11 per gli sprite attaccati
12    dc.w $1B8,$BBB  ; color28, COLORE 12 per gli sprite attaccati
13    dc.w $1BA,$8E0  ; color29, COLORE 13 per gli sprite attaccati
14    dc.w $1BC,$a70  ; color30, COLORE 14 per gli sprite attaccati
15    dc.w $1BE,$d00  ; color31, COLORE 15 per gli sprite attaccati
```

Per “attaccare” due sprite basta porre ad 1 il bit 7 della seconda word di controllo dello sprite dispari della coppia (ossia del famigerato quarto byte delle funzioni speciali). Per esempio per attaccare gli sprite 0 ed 1 basta settare tale bit allo sprite 1, per attaccare lo sprite 4 ed il 5 basta settarlo al 5. è ovvio che gli sprite attaccati devono avere le stesse coordinate, ossia essere l’uno sopra l’altro, per consentire la giusta sovrapposizione dei 4 piani. Facciamo un esempio: per attaccare gli sprite 0 ed 1 bisogna porre ad 1 il bit 7 del quarto byte dello sprite 1:

```
1  SPRITE0:
2  VSTART0:            ; posizione inizio VERTICALE
3      dc.b $50
4  HSTART0:            ; posizione inizio ORIZZONTALE
5      dc.b $90
6  VSTOP0:
7      dc.b $64        ; posizione fine VERTICALE
8      dc.b $00        ; non occorre settare il bit 7 agli sprite pari.
9  ; da qua iniziano i dati dei 2 piani dello sprite
10     dc.w %0000000000000000,%0000110000110000
11     dc.w %0000000000000000,%0000011001100000
12     ...
13     dc.w 0,0        ; fine sprite0
14
15 SPRITE1:
16 VSTART1:            ; posizione inizio VERTICALE
17     dc.b $50
18 HSTART:             ; posizione inizio ORIZZONTALE
```

```
19      dc.b $90
20  VSTOP:
21      dc.b $64        ; posizione fine VERTICALE
22
23            ;76543210
24      dc.b %10000000 ; BIT 7 SETTATO! ATTACCHED MODE per sprite 0/1
25
26  ; da qua iniziano i dati dei 2 piani dello sprite
27      dc.w %0000000000000000,%0000110000110000
28      dc.w %0000000000000000,%0000011001100000
29      ...
30      dc.w 0,0        ; fine sprite1
```

Dunque per far sì che tutti gli sprite siano in modo "attached" basta porre ad 1 i bit 7 del quarto byte degli sprite 1,3,5 e 7, cioè di quelli dispari.

Per farsi uno sprite a 16 colori è necessario disegnarlo con un programma da disegno e convertirlo in formato SPRITE con l'iffconverter KEFCON, infatti è difficile "calcolarsi" ad occhio i colori risultanti da 4 piani di bit, divisi in due sprite!

Caricatevi ed eseguitevi il listato `Lezione7g.s`, che visualizza uno sprite a 16 colori in modalità attached, in cui è anche descritto come convertirsi uno sprite con il KEFCON, sia a 4 colori che a 16.

È possibile visualizzare contemporaneamente sprite a 16 colori e sprite a 4 colori, per esempio gli sprite 0 ed 1 "attaccati" e gli altri no, o qualsiasi altra combinazione.

Nel listato esempio `Lezione7h.s` sono visualizzati i 4 sprite attached a 16 colori, ognuno con un movimento indipendente dagli altri.

A questo punto vi starete chiedendo come mai non è ancora stato eliminato l'inconveniente dello scorrimento orizzontale scattoso a passi di 2 pixel alla volta anziché uno. Ebbene, è giunta l'ora di risolvere il problema, ma per fare ciò è necessario imparare una nuova istruzione del 68000, la quale opera sui singoli bit di un numero: `LSR`.

Questa istruzione significa *Logic Shift Right*, cioè *scorrimento logico dei bit a destra*, in altre parole, se un numero binario in `d0` è %00111, dopo un bel `LSR #1,d0` il risultato è %00011, dopo un `LSR #2,d0` è %00001. Allo stesso modo, un %00110010 dopo un `LSR #1,d0` diventa %00011001, mentre dopo un `LSR #5,d0` diventa %00000001. Dunque il numero, considerato nella sua forma binaria, viene spostato a destra come se i bit fossero su una tovaglia che tiriamo: tirando di #1 si sposta la tovaglia con tutti i BitPiatti sopra e il primo BitPiatto cade in terra... tirando troppo si può spostare tutto facendo cadere tutto in terra e azzerando il tavolo.

Ma cosa c'entra questa istruzione assembler con il byte `HSTART`??? Il problema sta in questi termini: come sapete le posizioni orizzontali possibili sono ben più di $FF (255), per il solo fatto che lo schermo è largo 320 pixel. Per indicare un numero superiore a 255 (8 bit, dallo zero al sette), occorre aggiungere almeno un altro bit, il nono, detto bit 8, in tal modo anziché un massimo di %11111111 ($ff) si può avere un massimo di %111111111, ossia 511, che per l'`HSTART` va benissimo. Ma dove mettere questo bit?? Quei mattacchioni dei progettisti hanno pensato bene di metterlo nel famigerato quarto byte di controllo, quello che abbiamo già visto per attaccare gli sprite (il bit 7 di tale byte infatti serve ad attaccare gli sprite per farli a 16 colori).

Avendo altri 6 bit liberi per usi vari, decisero di usare il bit 0 come bit *basso* della coordinata a 9 bit della coordinata orizzontale, spezzando il numero a 9 bit in questo modo:

```
                    ; numero a 9 bit, che rappresenta la
      ;876543210    ; coordinata HSTART
      %111111111
       \_____/ \/
          |     |
     8 bit alti  |
     messi nel   |
```

```
byte HSTART |
            |
            |
            |
            |
            |
          bit 0 del
          numero a 9 bit
          messo nel bit 0
          del quarto byte
          di controllo
```

Se togliete il bit basso ad un numero a 9 bit, otterrete sempre numeri pari, dato che il bit 0 è sempre a zero. è quando il bit 0 è ad 1 che il numero è dispari, fate una prova con *<?100>* e *<?101>*, verificherete che i numeri pari hanno sempre il bit 0 azzerato, mentre quelli dispari hanno sempre il bit 0 settato. Fino ad ora dunque potevamo andare a scatti di 2 pixel per volta e dovevamo mettere la metà del valore effettivo in HSTART per questo motivo. Per poter raggiungere anche i numeri dispari ed inserire la vera coordinata in input, basterà dividere tale coordinata reale in bit basso e byte alto, poi mettere il bit basso al suo posto e il byte alto al suo posto. Per fare ciò immaginate di avere la coordinata dispari 35: (%00100011)

Quello che dobbiamo fare innanzitutto è controllare se il bit 0 del quarto byte di controllo va settato, e per fare questo basta testare se il numero in questione ha il bit 0 settato con un BTST, dopodiché si agirà di conseguenza: supponiamo di avere la coordinata in D0:

```
1         btst    #0,D0           ; bit basso della coordinata X azzerato?
2         beq.s   BitBassoZERO
3         bset    #0,MIOSPRITE+3  ; Settiamo il bit basso di HSTART
4         bra.s   PlaceCoords
5
6   BitBassoZERO:
7         bclr    #0,MIOSPRITE+3  ; Azzeriamo il bit basso di HSTART
8   PlaceCoords:
9         ....
```

Ora abbiamo settato o azzerato il bit basso di HSTART, non rimane che mettere il resto del numero, gli 8 bit più alti, nel byte HSTART come sempre. Ma c'è un problema: il numero ha 9 bit, e a noi servono solo gli 8 bit alti! A questo punto compare sulla scena l'istruzione LSR!!! Infatti assolve al compito di "far scorrere" i bit del numero di una posizione verso destra, facendo sparire il bit basso e posizionando gli 8 bit che ci servono al punto giusto, vediamo il seguito della routine sotto la label PlaceCoords:

```
1         lsr.w   #1,D0           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
2                                 ; il valore di HSTART, per "trasformarlo" nel
3                                 ; valore fa porre nel byte HSTART, senza cioè
4                                 ; il bit basso.
5         move.b  D0,HSTART       ; Poniamo il valore XX nel byte HSTART
6         rts
```

In questo caso avevamo la coordinata %00100011 (35), vediamo come diventa dopo l'LSR.W #1,d0: %00010001!!!! Ossia il byte giusto da mettere in HSTART.

In Lezione7i.s questa routine viene fatta funzionare per far scorrere uno sprite *finalmente* fluido come solo l'Amiga sa fare.

A questo punto possiamo eliminare anche l'ultima limitazione, ossia quella in senso verticale: infatti verticalmente possiamo spostare lo sprite a scatti di un pixel, ma solo fino alla linea $FF. I progettisti Amiga hanno optato per una soluzione diversa da quella dell'HSTART per VSTART/VSTOP: infatti anche VSTART e VSTOP necessitano di un numero a 9 bit anziché 8, ma anziché separare il bit più basso (lo zero) dagli altri 8 più alti, hanno separato il bit più alto (il nono) dagli otto più bassi, in questo modo in VSTART e VSTOP il numero è valido fino a $FF, ossia 255, dopodiché è necessario settare il nono bit che è stato messo nel famigerato quarto byte di

controllo, infatti dopo $ff viene $100, $101 eccetera, dunque il byte basso riparte da zero, ma col nono bit settato. Vediamo come fare una routine analoga a quella vista per la posizione orizzontale, ossia che parte dalla coordinata reale (è necessaria una word) e la "divide" in bit alto e byte basso. Da ricordare che in questo caso abbiamo da aggiornare anche VSTOP oltre a VSTART ogni volta!!! Teniamo presente che il bit alto di VSTOP è il bit 1 del quarto byte di controllo, mentre quello di VSTART è il bit 2:

```
        MOVE.w  (A0),d0         ; copia la word dalla tabella in d0
        ADD.W   #$2c,d0         ; aggiungi l'offset dell'inizio dello schermo
        MOVE.b  d0,VSTART       ; copia il byte in VSTART
        btst.l  #8,d0           ; numero maggiore di $FF?
        beq.s   NonVSTARTSET
        bset.b  #2,MIOSPRITE+3  ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,MIOSPRITE+3  ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   #13,D0          ; Aggiungi la lunghezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,VSTOP        ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,MIOSPRITE+3  ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,MIOSPRITE+3  ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:
        rts
```

Questa routine funziona in maniera analoga alla precedente per il settaggio del bit "staccato", mentre si differenzia per il fatto che deve agire sia su VSTART che su VSTOP, e per l'assenza dell'LSR, qua inutile. Potete provarla in pratica caricando la Lezione7l.s.

Ora che abbiamo il completo controllo sugli sprite, vediamo di ottimizzare le routine con le quali li controlliamo: innanzitutto, la prima cosa da fare è quella di fare una routine universale di controllo degli sprite, in modo da non dover riscrivere per ognuno degli 8 sprite la parte della sistemazione del bit "staccato". Serve una routine parametrica, la quale richieda in entrata l'indirizzo dello sprite interessato e la coordinata X ed Y che deve assumere, in questo modo basterà eseguire un BSR Routine per ogni sprite anziché riscrivere tutto. Potremo così riutilizzare tale routine ogni volta che vogliamo programmare gli sprite, al massimo con piccole modifiche. Un esempio di routine del genere la troviamo nella lezione7m.s.

La routine universale si chiama UniMuoviSprite, e per funzionare è necessario che le vengano indicate oltre all'indirizzo dello sprite da muovere e alle nuove coordinate che deve assumere, anche l'altezza dello sprite, che serve alla routine per calcolare il valore del byte VSTOP. Questi valori vengono comunicati o meglio "passati" alla routine mettendoli in alcuni registri prima di eseguire la routine.

Più precisamente si deve mettere l'indirizzo dello sprite nel registro a1, la sua altezza nel registro d2, la coordinata Y nel registro d0 e la coordinata X nel registro d1. La coordinate dello sprite "passate" alla routine sono i valori nello schermo 320x256. Infatti la routine si occupa di "centrare" lo sprite sullo schermo sommando $40 alla coordinata X e $2c alla coordinata Y. Inoltre pensa a mettere a posto il bit basso di HSTART e i bit alti di VSTART e VSTOP.

Brevemente:

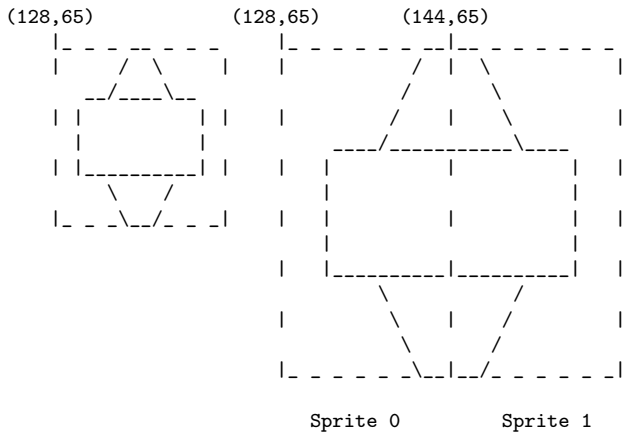
```
;
;       Parametri in entrata di UniMuoviSprite:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;       d2 = altezza dello sprite
;
```

Avendo a disposizione questa routine che ci risolve una volta per tutte i problemi relativi al posizionamento degli sprite, possiamo divertirci a usarla per qualche applicazione che ci consentirà di fare un po' di esperienza con gli sprite. Prima di proseguire però caricate ed eseguite la `Lezione7m.s`, e guai a voi se continuate a leggere la `lezione7.txt` o a caricare listati prima di averla capita **completamente**. Dato che sarà usata in tutti gli altri esempi sugli sprite, sarebbe poco produttivo continuare senza aver capito una routine che trovate continuamente.

Nella `lezione7n.s` vediamo uno sprite che si muove sullo schermo seguendo traiettorie rettilinee. Le posizioni dello sprite non sono contenute in un tabella, ma vengono calcolate di volta in volta facendo muovere lo sprite con velocità costante. Caricatelo ed eseguitelo, vedremo anche come far rimbalzare uno sprite contro i bordi dello schermo.

Nella `lezione7o.s` vedremo invece due sprite che vengono entrambi mossi dalla routine universale. È un ottimo esempio di come grazie all'utilizzo dei parametri la nostra routine universale sia in grado di muovere senza nessuna modifica sprite che hanno forma e dimensioni diverse. Se non usassimo i parametri dovremmo scrivere una routine per ciascun sprite, sprecando tempo per farlo e memoria sul computer (con 8 sprite dovremmo scrivere 8 routine).

Nella `lezione7p.s` sempre usando la routine universale vediamo come si possano creare degli oggetti larghi più di 16 pixel utilizzando degli sprite affiancati. State *bene attenti* a non confondere gli sprite "attaccati" con quelli "affiancati": i primi sono 2 sprite della stessa coppia usati in modalità "attached", hanno le stesse coordinate (sono perfettamente sovrapposti) e lo sprite dispari ha il bit "attach" settato a 1; per sprite "affiancati" si intende invece un insieme di due o più sprite che vengono posizionati sullo schermo uno di fianco all'altro senza lasciare neppure una colonna di pixel tra l'uno e l'altro, in maniera da sembrare un unico oggetto largo più di 16 pixel, dato che sono mossi contemporaneamente. Non c'è nessun bit da settare per gli sprite affiancati, non si tratta di un "modo" speciale degli sprite, ma solo di una particolare disposizione sullo schermo di normalissimi sprite. Ecco uno schemino che mostra un'astronave fatta da un solo sprite, e un'altra fatta da due sprites:



Con una tecnica del genere si possono fare mostri di fine livello larghi fino a 128 pixel (16*8) se fatti con sprite a 3 colori, oppure larghi fino a 64 pixel (16*4) se fatti con sprite attached a 15 colori. Se il mostro in questione è più alto che largo, ad esempio di forma umana, si potrebbe sfruttare tutta la lunghezza dello schermo, dato che non ci sono limiti per l'altezza di uno sprite, e si potrebbe cambiare la palette verticalmente col copper per colorargli, ad esempio, le scarpe con un colore diverso dai jeans.

7.4 Mouse e joystick

Ora che abbiamo visto come far muovere gli sprite all'Amiga, perché non impariamo muoverli noi? Naturalmente con l'aiuto di un joystick o di un mouse!

Prima di vedere come si usano questi dispositivi è necessario imparare delle nuove istruzioni assembler, che riguardano la manipolazione dei bit di un registro e che si chiamano `NOT`, `AND`, `OR`, `EOR`. Queste istruzioni lavorano sui singoli bit di un registro (o di una locazione di memoria), sia per il registro sorgente che per quello destinazione. Ad esempio queste istruzioni considerano un byte non come un numero formato da 8 bit (cifre binarie) ma come un insieme di 8 bit indipendenti tra di loro. In pratica questo vuol dire che l'effetto che l'istruzione ha su un singolo bit del registro è indipendente da quello che succede agli altri bit del registro.

Per prima vediamo la `NOT`. Essa funziona su un solo operando, e il suo effetto è quello di rovesciare i bit dell'operando, cioè di scambiare 1 con 0 e 0 con 1. Se ad esempio nel registro d0 abbiamo il numero %01001100, se facciamo

```
1       NOT.B   d0
```

il risultato sarà %10110011.

Le altre 3 istruzioni, invece lavorano con 2 operandi, uno sorgente e l'altro destinazione, fanno un'operazione tra i contenuti degli operandi e mettono il risultato nell'operando destinazione. Le operazioni (che sono ovviamente diverse per ogni istruzione) sono bit-a-bit, cioè avvengono tra ogni bit dell'operando sorgente e il corrispondente bit dell'operando destinazione, nel quale inoltre viene poi messo il risultato. Quindi fare `D0 AND D1` in pratica significa fare:

```
(bit 0 di D0) AND (bit 0 di D1)
(bit 1 di D0) AND (bit 1 di D1)
(bit 2 di D0) AND (bit 2 di D1) e così via per tutti i bit di D0 e D1
```

Vediamo dunque come funziona l'`AND` tra 2 bit. Poiché un bit vale 0 o 1, ci sono 4 possibili casi:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

`AND` da come risultato 1 soltanto quando sia il bit del primo operando che quello del secondo operando sono ad 1. Infatti `AND` in inglese significa "e", quindi da come risultato 1 se il primo E il secondo bit sono a 1. Si potrebbe tradurre con: *sono ad 1 sia il primo che il secondo bit? Se si, rispondo con 1, se no invece rispondo con uno zero*. Un `AND` può essere utile ad azzerare certi bit di un numero:

```
1   AND.W #%1111111111111011,LABEL
```

Azzererà il bit 2 del numero in `LABEL`, perché è l'unico che è azzerato nell'operando, e l'unico che sarà cambiato nella destinazione, infatti tutti gli altri sono ad 1, quindi questi non cambiano la destinazione. Se il bit di destinazione è 0, facendo un `1 AND 0` il risultato rimane 0, allo stesso modo se è 1, facendo un `1 AND 1` il risultato rimane 1. Per quanto riguarda il bit che è a 0, invece, condanna la destinazione ad essere 0, infatti per dare un 1 di destinazione entrambi gli operandi devono essere 1, in questo caso essendo a 0 il primo, sia che il secondo sia 0 o 1 il risultato sarà 0. alcuni esempi:

```
1111001111 AND 0011001100 = 0011001100 - Nessuna modifica
1101011011 AND 0001110001 = 0001010001 - 1 bit azzerato
1111101101 AND 0011111111 = 0011101101 - 2 bit azzerati
```

Questa operazione di azzeramento si dice *mascheratura*:

```
AND #%11110000,LABEL    (%11110000 è la maschera, infatti è come se si
                         mettesse una maschera di ZERI sopra il numero
                         in LABEL, in questo caso è come se "tappassimo"
                         i primi 4 bit come si "tappa" un neo una ragazza
                         quando si mette il fondotinta. Il neo è un 1 che
                         si trovava nella posizione della maschera dove
                         c'erano degli 0, e il neo che viene "coperto"
                         dal trucco, ossia viene azzerato).
```

L'OR invece si comporta in questo modo:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

In questo caso basta che 1 dei 2 bit sia ad 1 per dare risultato 1. Dunque il risultato è sempre 1 tranne quando entrambi i bit sono a zero. Anche qui sapere che OR in inglese significa "o" ci aiuta a ricordare che il risultato e' 1 se il primo O il secondo bit sono a 1. Si potrebbe tradurre in *o uno o l'altro bit devono essere ad 1 per dare 1*. Questo comando è utile, all'opposto dell'AND, per *settare* dei bit, per porli cioè ad 1: alcuni esempi:

```
0000000001 OR 1101011101 = 1101010001 - Nessun cambiamento
1000000000 OR 0010011000 = 1010011000 - 1 bit settato
0001111000 OR 1111100000 = 1111111000 - 2 bit settati
```

In questo caso, è come se la ragazza di prima, anziché mettersi il fondotinta rosaceo (gli 0) per tappare i nei neri (ossia gli 1), si mettesse del nero per farsi dei nei falsi, come quello che aveva Marilin Monroe sopra il labbro. Oppure come se fosse una ragazza di colore (ossia tutta ad 1) che si è truccata con il rosa per sembrare bianca (come Michael Jackson), ossia per essere tutta a zero, che si toglie il fondotinta dove il numero dell'OR è ad uno, scoprendo il nero.

Invece il comando EOR, ovvero OR esclusivo, setta il bit solo quando è ad 1 o il primo o il secondo bit, non quando sono ad 1 entrambi, come invece fa il comando OR:

```
0 EOR 0 = 0
0 EOR 1 = 1
1 EOR 0 = 1
1 EOR 1 = 0  ; Questa è la differenza con l'OR! infatti 1 OR 1 = 1.
```

Alcuni esempi:

```
0000000001 EOR 1101011101 = 1101010000 - 1 bit azzerato
1000000000 EOR 0010011000 = 1010011000 - 1 bit settato
```

Quest'ultima istruzione ci sarà utile per leggere il joystick. Come sapete l'Amiga ha 2 porte usate per collegare joystick o mouse. Ad ognuna di queste porte si può collegare indifferentemente un joystick o un mouse. Per ogni porta esiste un registro hardware che si può leggere per sapere se e in che modo sono mossi joystick e mouse. La porta 0 (dove di solito è collegato il mouse) viene letta attraverso il registro JOY0DAT ($dff00a) mentre la porta 1 attraverso JOY1DAT ($dff00c).

Per prima cosa vediamo come leggere il joystick. Ci riferiremo al registro JOY1DAT che è quello usato di solito, ma JOY0DAT funziona esattamente allo stesso modo quando ci colleghiamo un joystick. Possiamo pensare ad un joystick come ad un insieme di 4 interruttori (uno per ogni direzione), ognuno dei quali può assumere 2 stati: chiuso (1) o aperto (0) a seconda che la leva del joystick sia premuta o meno nella direzione associata all'interruttore. Per sapere in quali direzioni è mosso il joystick dobbiamo conoscere gli stati degli interruttori. Per 2 di questi interruttori è molto semplice, in quanto il loro stato è riportato in un bit del registro JOY1DAT:

- il bit 1 di JOY1DAT è lo stato dell'interruttore "destra"
- il bit 9 di JOY1DAT è lo stato dell'interruttore "sinistra".

Se un bit vale 1 l'interruttore associato è chiuso, altrimenti è aperto. Per quanto riguarda le altre 2 direzioni lo stato non è mappato direttamente in un bit, ma deve essere ottenuto mediante il calcolo di un'operazione, precisamente dell'EOR che abbiamo spiegato poco fa, effettuata tra 2 bit del registro JOY1DAT:

- lo stato dell'interruttore "alto" è il risultato di un EOR tra il bit 8 e il bit 9
- lo stato dell'interruttore "basso" è il risultato di un EOR tra il bit 0 e il bit 1.

Anche in questo caso se un bit vale 1 l'interruttore associato è chiuso, altrimenti è aperto. Conoscendo gli stati dei 4 interruttori possiamo dunque usare il joystick per muovere uno sprite sullo schermo.

Caricate in un altro buffer di testo la lezione7q.s ed eseguitela. Veniamo ora al mouse. Quando colleghiamo un mouse ad una delle porte, il registro corrispondente si comporta in maniera diversa che nel caso del joystick. Infatti prendendo il registro JOY0DAT (ma è lo stesso per l'1), troviamo che il byte alto è usato per rilevare gli spostamenti in direzione verticale e quello basso quelli in direzione orizzontale. Ogni byte rappresenta un numero (da 0 a 255) che varia secondo i movimenti del mouse.

- il byte alto diminuisce ogni volta che il mouse viene spostato verso l'alto e aumenta ogni volta che il mouse viene spostato verso il basso.
- il byte basso diminuisce ogni volta che il mouse viene spostato verso sinistra e aumenta ogni volta che il mouse viene spostato verso destra.

Vediamo come usare queste informazioni per muovere uno sprite con il mouse.

Il primo metodo che viene in mente è di usare i 2 byte di JOY0DAT come coordinate per lo sprite, visto che anche le coordinate dello sprite diminuiscono se esso va in alto o a sinistra e aumentano se va in basso o a destra. Questo metodo ha l'inconveniente che in un byte possiamo raggiungere il valore 255, quindi i valori che possiamo leggere dal byte di JOY0DAT dedicato alla direzione orizzontale possono arrivare al massimo a 255, mentre le coordinate orizzontali di uno sprite possono arrivare oltre 320. Caricate Lezione7r1.s e verificate questo metodo.

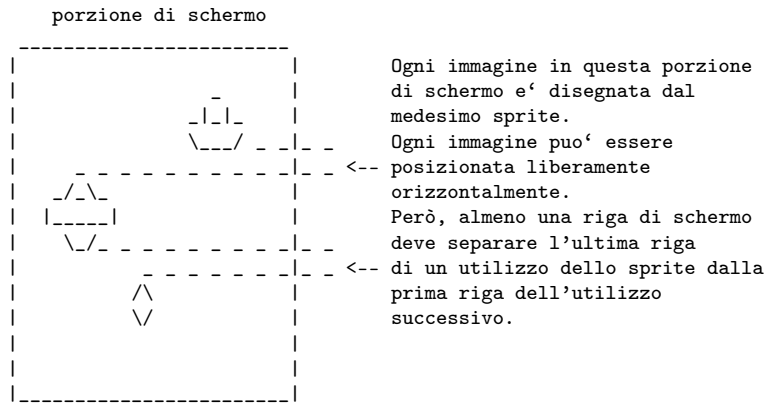
Un metodo un pò più complesso che però risolve il problema della limitazione in senso orizzontale a 255 pixel anziché 320 è presentato in Lezione7r2.s. Per una spiegazione del metodo leggetevi il commento alla fine del listato.

Sapendo come muovere una freccia sullo schermo, si può facilmente simulare il sistema intuition, ossia si può fare un pannello di controllo con dei bottoni disegnati da attivare spostandoci la freccia (lo sprite) sopra e premendo il pulsante, sia esso del joystick o del mouse. Basta controllare al momento della pressione del bottone in quale coordinata si trova la freccia, e se si trova sopra un bottone attivare l'opzione di quel bottone. Fare questo è piuttosto facile, provate da voi a farlo. Comunque in lezioni più avanzate del corso ci sarà un listato di questo tipo.

7.5 Riutilizzo degli sprite

Il riutilizzo degli sprite è una tecnica che ci consente di visualizzare più di 8 sprite contemporaneamente. In pratica uno stesso sprite viene usato per disegnare diversi oggetti situati a diverse altezze. Se ad esempio utilizziamo uno sprite per visualizzare un alieno nella parte alta dello

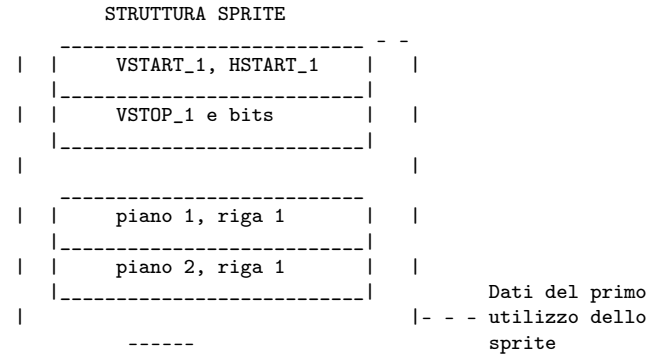
schermo, possiamo poi utilizzare di nuovo lo stesso sprite per disegnare l'astronave del giocatore nella parte bassa dello schermo. L'unica limitazione che si ha quando si riutilizzano gli sprite è che 2 oggetti disegnati da uno stesso sprite devono essere posizionati ad altezze differenti. Non è possibile visualizzare su una stessa riga dello schermo 2 righe che compongono 2 oggetti disegnati con il medesimo sprite. Per di più l'ultima riga della figura disegnata durante un utilizzo e la prima riga della figura disegnata con l'utilizzo successivo dello stesso sprite *devono* essere separate da almeno una riga nella quale lo sprite non è utilizzato. La figura seguente illustra meglio la situazione:

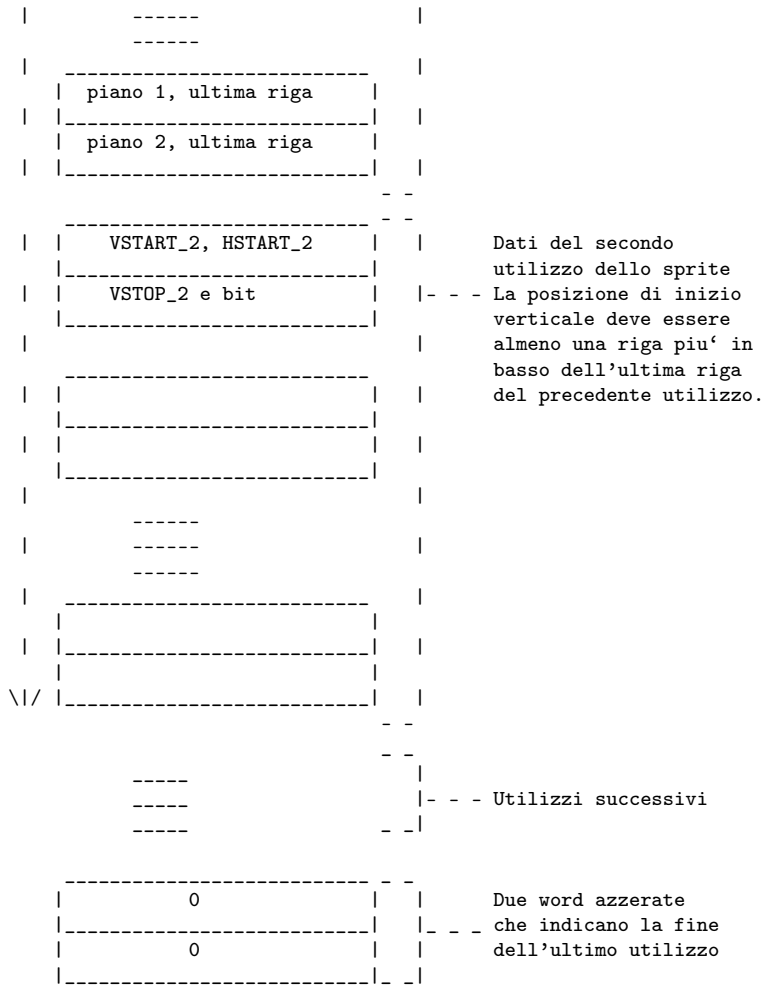


Non c'è nessuna limitazione invece per quanto riguarda le posizioni orizzontali, né per quanto riguarda figure disegnate mediante sprites diversi. Uno sprite può essere riutilizzato un numero qualunque di volte, ogni volta ad un'altezza diversa. Questa tecnica si può applicare ad ogni sprite, e in modo indipendente tra uno sprite e l'altro: per esempio si possono utilizzare 1 volta sola gli sprite 0,3 e 4, utilizzare 3 volte lo sprite 1, quattro volte lo sprite 2 e non utilizzare affatto gli sprite 5,6 e 7.

Applicare questa tecnica è molto semplice, in quanto richiede solo una modifica della struttura dati dello sprite.

Normalmente, alla fine della struttura dello sprite, dopo tutti i dati che descrivono la forma ci sono 2 word di valore 0 che appunto indicano la fine della struttura. Per riutilizzare uno sprite, al posto di queste 2 word ci mettiamo un'altra struttura sprite, che descrive un'altra figura da disegnare sullo schermo più in basso della prima. Se si vuole riutilizzare per una terza volta lo sprite, si mette una terza struttura sprite subito dopo la seconda, e lo stesso si fa per tutti i riutilizzi che si vuole. Dopo la struttura dati dell'ultimo utilizzo si mettono le 2 word di valore 0 che indicano la fine dell'ultimo utilizzo.





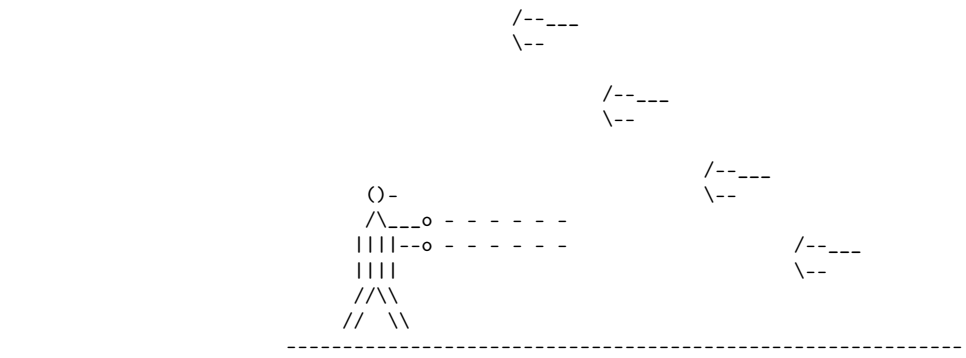
Da notare che i vari utilizzi verticali devono essere messi nella struttura in ordine da quello più in alto a quello più in basso. Per cui il byte `VSTART` di ogni utilizzo deve essere *maggiore* del byte `VSTOP` dell'utilizzo precedente dello sprite.

Vediamo un esempio pratico di struttura in cui uno sprite è riutilizzato 2 volte:

```
1  MIOSPRITE:
2  VSTART_1:
3          dc.b $50                                ; posizione primo utilizzo
4  HSTART_1:
5          dc.b $40+12
6  VSTOP_1:
7          dc.b $58
8          dc.b $00
9   dc.w   %0000001111000000,%0111110000111110     ; dati "forma" del primo
10  dc.w   %0000111111110000,%1111001110001111     ; utilizzo
11  dc.w   %0011111111111100,%1100010001000011
12  dc.w   %0111111111111110,%1000010001000001
13  dc.w   %0111111111111110,%1000010001000001
14  dc.w   %0011111111111100,%1100010001000011
15  dc.w   %0000111111110000,%1111001110001111
16  dc.w   %0000001111000000,%0111110000111110
17 VSTART_2:                                       ; posizione utilizzo 2
18         dc.b $70                          ; NOTATE CHE VSTART_2 > VSTOP_1
19 HSTART_2:
```

```
20          dc.b $40+20
21  VSTOP_2:
22          dc.b $78
23          dc.b $00
24   dc.w   %0000001111000000,%0111110000111110      ; dati "forma" del
25   dc.w   %0000111111110000,%1111001110001111      ; secondo utilizzo
26   dc.w   %0011111111111100,%1100010001000011
27   dc.w   %0111111111111110,%1000001110000001
28   dc.w   %0111111111111110,%1000010001000001
29   dc.w   %0011111111111100,%1100010001000011
30   dc.w   %0000111111110000,%1111001110001111
31   dc.w   %0000001111000000,%0111110000111110
32   dc.w   0,0                                     ; fine ultimo utilizzo
```

La tecnica del riutilizzo, se ben sfruttata, può portare a moltiplicare gli oggetti in movimento in uno shoot'em'up. Ad esempio in un gioco a scrolling orizzontale, dove i nemici si muovono orizzontalmente:



La formazione nemica, essendo formata da oggetti che si spostano in senso per lo più orizzontale, senza mai andare l'uno sopra l'altro, può essere fatta da un solo sprite riutilizzato. In questo modo avremmo altri 7 sprite per il player1 e per le eventuali bombe.

Un esempio di utilizzo di questa tecnica lo potete trovare in `lezione7s.s` dove visualizzeremo "16" sprite contemporaneamente. Caricatelo e studiatevelo.

Non poteva poi mancare nel nostro corso, uno degli effetti più "classici" delle intro di qualche anno fa: lo "starfield", ovvero le stelle che si muovono orizzontalmente. Le stelle infatti sono realizzate usando uno sprite riutilizzato. Ne presentiamo 3 versioni in `lezione7t1.s`, `lezione7t2.s` e `lezione7t3.s`. Il riutilizzo degli sprite si può applicare anche agli sprite "attaccati", allo stesso modo degli sprite normali. In `lezione7t4.s` vediamo un esempio nel quale realizziamo un effetto simile allo "starfield" ma con palline colorate al posto delle stelle.

7.6 Il dual playfield mode

Prima di spiegare altre caratteristiche degli sprite, faremo una digressione per approfondire la trattazione del Dual Playfield mode. Come abbiamo già accennato nella lezione 4, il Dual Playfield è un modo grafico speciale che consente di visualizzare due schermi sovrapposti, chiamati PLAYFIELD 1 e 2. Cosa vuol dire che i 2 schermi sono sovrapposti? In pratica ogni playfield, ha un colore "trasparente" attraverso il quale si può vedere quello che c'è sotto, esattamente allo stesso modo del colore 0 di ogni sprite. In pratica il trasparente non è un vero colore ma una sorta di "buco" all'interno del playfield. Gli altri colori di ogni playfield si comportano invece normalmente. Uno dei due *playfield* (a scelta) appare al di sopra dell'altro, e i suoi colori *non* trasparenti ricoprono l'altro playfield; il trasparente, invece si comporta come un buco e lascia vedere il playfield sottostante.

Il numero massimo di bit-planes che ognuno dei 2 playfield può avere è 3 bit-plane in LOW-RES e 2 bit-plane in HI-RES. In pratica i 6 bit-planes dell'Amiga vengono ripartiti in due gruppi da 3, e ogni gruppo costituisce un playfield. Il playfield 1 è costituito dai bit-plane dispari, cioè i bit-plane 1, 3 e 5. Il playfield 2 è costituito dai bit-plane pari, cioè 2, 4 e 6.

Naturalmente non è sempre necessario usare tutti i bit-plane disponibili. Non possiamo però assegnare indipendentemente ai 2 playfield i bit-plane che vogliamo. Infatti il numero di bit-planes da usare si indica esattamente nello stesso modo che per i modi grafici "normali". Nei bit 14-12 del registro `BPLCON0` (`$dff100`), chiamati bit `BPU2`, `BPU1` e `BPU0` viene indicato il numero complessivo di bit-plane da attivare nei 2 playfield. In base al numero complessivo che noi indichiamo nei bit `BPU`, l'hardware assegna i bit plane secondo la seguente tabella:

Numero di bitplanes usati (bit BPU di BPLCON0)	Bit-planes al Playfield 1	Bit-planes al Playfield 2
0	nessuno	nessuno
1	plane 1	nessuno
2	plane 1	plane 2
3	plane 1,3	plane 2
4	plane 1,3	plane 2,4
5	plane 1,3,5	plane 2,4
6	plane 1,3,5	plane 2,4,6

Come potete vedere in pratica il playfield 1 ha sempre più planes del playfield 2, ed inoltre, il playfield 2 ha al massimo un plane in meno del playfield 1; non è possibile assegnare 3 plane al playfield 1 e un solo plane al playfield 2.

Analogamente ai modi grafici standard, la sovrapposizione dei bit-plane determina il colore usato per rappresentare ogni pixel sul video. Però la corrispondenza tra combinazioni dei bit-plane e registri colore è un pò differente, ed illustrata nelle 2 tabelle seguenti:

PLAYFIELD 1

Valore plane 5	Valore plane 3	Valore plane 1	Colore selezionato
0	0	0	trasparente
0	0	1	COLOR01
0	1	0	COLOR02
0	1	1	COLOR03
1	0	0	COLOR04
1	0	1	COLOR05
1	1	0	COLOR06
1	1	1	COLOR07

```
PLAYFIELD 2
  Valore        |  Valore       |  Valore       |  Colore
        plane 6 |        plane 4 |       plane 2 |  selezionato
----------------------------------------------------
                |               |               |
        0       |       0       |       0       | trasparente
                |               |               |
        0       |       0       |       1       |  COLOR09
                |               |               |
        0       |       1       |       0       |  COLOR10
                |               |               |
        0       |       1       |       1       |  COLOR11
                |               |               |
        1       |       0       |       0       |  COLOR12
                |               |               |
        1       |       0       |       1       |  COLOR13
                |               |               |
        1       |       1       |       0       |  COLOR14
                |               |               |
        1       |       1       |       1       |  COLOR15
```

A questo punto sapete come funziona il Dual Playfield mode. C'è solo unica cosa che non sapete... come si attiva il dual playfield !! È molto semplice basta settare a 1 il bit 10 del registro `BPLCON0`. Come abbiamo già detto è possibile scegliere quale dei due playfield appaia al di sopra dell'altro. Si dice che il playfield che appare sopra ha priorità maggiore. C'è un bit che determina la priorità, il bit 6 del registro `BPLCON2` (`$dff104`): se esso vale 0 il playfield 1 appare sopra al 2, se invece vale 1 è il playfield 2 che appare sopra all'1. Potete vedere un esempio di Dual Playfield in lezione7u.s

7.7 Priorità tra sprite e playfield

Abbiamo già visto le priorità relative dei vari sprite. Cioè se due sprite si sovrappongono, quello con il numero più basso apparirà al di sopra dell'altro. Inoltre abbiamo appena visto come stabilire la priorita' tra i 2 playfield nel modo Dual Playfield. Non ci resta ora che vedere le priorità tra sprite e playfield. Innanzitutto notiamo che gli sprite appaiono sempre al di sopra del colore zero. Per gli altri colori la priorità è controllata dal registro `BPLCON2`. È possibile settare la priorità indipendentemente per i bit-planes pari e per i dispari. Ciò è molto utile nel modo Dual Playfield, perché ci consente di dare ad ogni playfied una diversa priorità rispetto agli sprite. Nel modo standard, invece è opportuno dare la stessa priorità rispetto agli sprite a planes pari e dispari. Il registro `BPLCON2` possiede alcuni bit nei quali scrivere il livello di priorità desiderato per planes pari e dispari. I bit da 0 a 2 contengono il livello di priorità dei bit-planes dispari (che corrispondono al PLAYFIELD 1 nel modo Dual Playfield) mentre i bit da 3 a 5 contengono il livello di priorità dei bit-planes pari (PLAYFIELD 2 nel modo Dual Playfield).

Vediamo come è codificato il livello di priorità, riferendoci ad un generico playfield, visto che la codifica è identica nei 2 casi. Per quanto riguarda le priorità con i playfield gli sprite si considerano a coppie (0-1, 2-3, 4-5 e 6-7). Come sappiamo, la priorità tra gli sprite (e quindi tra le coppie) è fissa:

```
PRIORITA' MASSIMA       COPPIA 1 (SPRITES 0 E 1)
                        COPPIA 2 (SPRITES 2 E 3)
                        COPPIA 3 (SPRITES 4 E 5)
PRIORITA' MINIMA        COPPIA 4 (SPRITES 6 E 7)
```

Il livello di priorità ci permette di inserire in questa pila il nostro playfield: lo possiamo mettere al di sopra di tutte le coppie, al di sotto di tutte le coppie, o in mezzo a 2 coppie. Non

è quindi possibile far comparire il playfield al di sotto della coppia 4 e al di sopra della coppia 2, perché la coppia 2 si trova più in alto della coppia 4 nella pila. È invece possibile il contrario. Mostriamo ora una tabella con tutte le possibili priorità, a seconda del livello che settiamo nei bit di BPLCON2

```
CODICE    |    000    |    001    |    010    |    011    |    100    |
-------------------------------------------------------------------------
PRI. MAX  | PLAYFIELD | COPPIA 1  | COPPIA 1  | COPPIA 1  | COPPIA 1  |
          | COPPIA 1  | PLAYFIELD | COPPIA 2  | COPPIA 2  | COPPIA 2  |
          | COPPIA 2  | COPPIA 2  | PLAYFIELD | COPPIA 3  | COPPIA 3  |
          | COPPIA 3  | COPPIA 3  | COPPIA 3  | PLAYFIELD | COPPIA 4  |
PRI. MIN  | COPPIA 4  | COPPIA 4  | COPPIA 4  | COPPIA 4  | PLAYFIELD |
```

Per esempio, come si vede dalla tabella, se vogliamo che gli sprite 0,1,2,3 (cioè le coppie 1 e 2) appaiano al di sopra del playfield e gli altri sprite invece al di sotto, dobbiamo scegliere il codice %010. Questo codice andrà scritto nel registro BPLCON2, nei bit da 0 a 2 se ci si riferisce al playfield 1 in dual-playfield, nei bit da 3 a 5 se ci si riferisce al playfield 2 in dual-playfield, mentre se stiamo usando uno schermo normale,lo dovremo scrivere 2 volte, sia nei bit da 0 a 2 che nei bit da 3 a 5.

In `lezione7v1.s` trovate un esempio di come settare le priorità degli sprite con uno schermo "normale". In `lezione7v2.s` invece, viene usato uno schermo Dual Playfield.

7.8 Collisioni

L'hardware di Amiga mette a disposizione del programmatore un sistema di rilevamento delle collisioni tra sprite e sprite, di quelle tra sprite e playfield e di quelle tra i 2 playfield. Tutti questi tipi di collisione vengono gestiti mediante 2 soli registri: CLXDAT (`$dff00e`) che è un registro a sola lettura nel quale vengono segnalate le collisioni, e CLXCON (`$dff098`), che è un registro di controllo mediante il quale si può modificare il modo in cui le collisioni vengono rilevate. Cominciamo illustrando la struttura di questi registri. I bit del registro CLXDAT si comportano come dei rilevatori di collisione. Ogni bit è dedicato ad un particolare tipo di collisione. Quando si verifica una collisione di un determinato tipo, il bit ad essa dedicato in CLXDAT assume il valore 1. Quando la collisione non si verifica più il bit ritorna al valore 0. Nella seguente tabella illustriamo il significato dei bit di CLXDAT:

```
USO DEI BIT DI CLXDAT

bit 15  non usato
bit 14  collisione tra coppia 3 e coppia 4
bit 13  collisione tra coppia 2 e coppia 4
bit 12  collisione tra coppia 2 e coppia 3
bit 11  collisione tra coppia 1 e coppia 4
bit 10  collisione tra coppia 1 e coppia 3
bit 9  collisione tra coppia 1 e coppia 2
bit 8  collisione tra playfield 2 e coppia 4
bit 7  collisione tra playfield 2 e coppia 3
bit 6  collisione tra playfield 2 e coppia 2
bit 5  collisione tra playfield 2 e coppia 1
bit 4  collisione tra playfield 1 e coppia 4
bit 3  collisione tra playfield 1 e coppia 3
bit 2  collisione tra playfield 1 e coppia 2
bit 1  collisione tra playfield 1 e coppia 1
bit 0  collisione tra playfield 1 e playfield 2
```

Il registro CLXCON ha la seguente struttura

```
USO BIT DI CLXCON
bit 15  abilita sprite 7
bit 14  abilita sprite 5
bit 13  abilita sprite 3
bit 12  abilita sprite 1
bit 11  abilita bit-plane 6
bit 10  abilita bit-plane 5
bit 9  abilita bit-plane 4
bit 8  abilita bit-plane 3
bit 7  abilita bit-plane 2
bit 6  abilita bit-plane 1
bit 5  valore-collisione bit-plane 6
bit 4  valore-collisione bit-plane 5
bit 3  valore-collisione bit-plane 4
bit 2  valore-collisione bit-plane 3
bit 1  valore-collisione bit-plane 2
bit 0  valore-collisione bit-plane 1
```

(nota: dove è scritto “abilita” si intende *abilita per il rilevamento collisioni*: se per esempio il bit 15 di CLXCON vale 0 *non* vuol dire che lo sprite 7 non può apparire sullo schermo, ma solo che le collisioni che riguardano lo sprite 7 non vengono rilevate)

Spiegheremo un po’ alla volta il significato di questi bit. Cominciamo a parlare della collisione tra sprite e sprite. Diciamo subito che anche per quanto riguarda le collisioni gli sprite sono considerati al livello di coppie. Infatti è possibile rilevare solo le collisioni tra sprite appartenenti a coppie diverse, e non fra sprite appartenenti alla stessa coppia. Per esempio non è possibile rilevare la collisione di sprite 0 con sprite 1. Invece vengono rilevate collisioni tra sprite appartenenti a coppie diverse. Per esempio se si verifica una collisione tra sprite 0 e sprite 2 il bit 9 di CLXDAT (collisione tra coppia 1 e coppia 2) assume il valore 1. Se si verifica una collisione tra sprite 1 e sprite 2, poiché anche lo sprite 1 appartiene (come lo 0) alla coppia 1, sarà sempre il bit 9 ad assumere il valore 1.

Questo però non accadrà sempre. Infatti le collisioni che riguardano gli sprite di numero pari (cioè gli sprite 0,2,4 e 6) vengono sempre rilevate, ma le collisioni che riguardano gli sprite dispari, vengono rilevate solo se noi vogliamo. Per abilitare uno sprite dispari al rilevamento collisioni, dobbiamo mettere a 1 il corrispondente bit di abilitazione nel registro CLXCON. Potete vedere quali sono i bit nella tabella che abbiamo riportato sopra. Gli sprite dispari possono essere abilitati indipendentemente l’uno dall’altro. Abilitare uno o più sprite dispari al rilevamento delle collisioni, comporta vantaggi e svantaggi.

Consideriamo per esempio solo le coppie 1 e 2, e supponiamo di non aver abilitato né lo sprite 1 né lo sprite 3. In questo caso, se si verifica una collisione tra sprite 0 e 2, il bit 9 (di CLXDAT) assume valore 1. Se invece la collisione si verifica tra sprite 1 e 2, oppure tra 0 e 3, oppure tra 1 e 3, non accade nulla, e noi non possiamo sapere che la collisione è avvenuta. Supponiamo invece di aver abilitato uno degli sprite dispari, per esempio sprite 1. In questo caso le collisioni tra sprite 0 e 2 e tra sprite 1 e 2 pongono a 1 il bit 9 di CLXDAT, mentre le collisioni tra sprite 0 e 3 e tra sprite 2 e 3 non provocano nessun effetto. In questa situazione c’è uno svantaggio rispetto al caso precedente, in cui lo sprite 1 non era abilitato. Infatti nel caso precedente, se il bit 9 assumeva il valore 1 eravamo sicuri che la collisione era avvenuta tra sprite 0 e sprite 2. Nel caso presente invece ci sono 2 possibilità: o c’è collisione tra sprite 0 e 2 oppure tra sprite 1 e 2. Non vi è modo di risolvere l’enigma leggendo il registro CLXDAT. Se lo sprite 1 è disabilitato, ma lo sprite 3 è abilitato, si ha una situazione analoga, in quanto vengono rilevate le collisioni tra sprite 0 e 2 e tra sprite 0 e 3, ma non si riesce a distinguere quale delle 2 si è verificata.

Infine nel caso in cui sono abilitati sia lo sprite 1 che il 3, vengono rilevate collisioni tra sprite 0 e 2, tra sprite 0 e 3, tra sprite 1 e 2 e tra sprite 1 e 3, e non c’è modo di distinguere.

Un esempio di collisione tra sprite, con gli sprite dispari disabilitati al rilevamento collisioni, è in `lezione7w1.s`. Caricatelo e verificatene il funzionamento.

Un esempio di collisione tra sprite con uno sprite dispari abilitato è nella `lezione7w2.s`. Noterete che questo esempio così com'è non funziona; per farlo funzionare, dovrete seguire le modifiche indicate nel commento. In questo esempio, per distinguere se una collisione riguarda lo sprite dispari abilitato, o lo sprite pari ad esso accoppiato, viene impiegata una tecnica basata sul confronto delle posizioni, illustrata nel commento.

Veniamo ora alla collisione tra sprite e playfield. È possibile rilevare una collisione tra una coppia di sprite e uno o più colori del playfield. Anche in questo caso le collisioni sono rivelate considerando coppie di sprite e non i singoli membri della coppia. L'abilitazione degli sprite dispari tramite i bit del registro `CLXCON` ha effetto anche in questo caso.

Il rilevamento delle collisioni avviene in modo diverso se stiamo usando uno schermo normale o Dual Playfield. Con uno schermo normale, i bit da 1 a 4 di `CLXDAT` indicano una collisione tra una coppia di sprite e il colore (o i colori) che abbiamo scelto per la collisione. Il bit 1 indica collisione tra il playfield e la coppia 1, il bit 2 tra playfield e coppia 2, il bit 3 tra playfield e coppia 3, il bit 4 tra playfield e coppia 4. I bit da 5 a 8 invece non vanno usati.

Nel caso di modo dual playfield è possibile rilevare una collisione tra uno dei 2 playfield e una coppia di sprite, e i bit di `CLXDAT` si usano come indicato nella tabella del registro `CLXDAT`: i bit da 1 a 4 indicano le collisioni tra playfield 1 e le varie coppie di sprite, mentre i bit da 5 a 8 indicano le collisioni tra playfield 2 e le varie coppie di sprite. Per scegliere i colori con cui rilevare le collisioni si usa il registro `CLXCON`. Iniziamo con il caso di un solo colore. I bit da 6 a 11 di CLXCON indicano quali bit-planes sono attivi per le collisioni. Nel caso in cui vogliamo rilevare collisioni tra sprite e un solo colore dobbiamo abilitare per le collisioni tutti i bit-planes che vengono visualizzati. La scelta del colore con cui rilevare una collisione viene effettuata scrivendo il numero del registro in cui è contenuto il colore nei bit da 0 a 5 di `CLXCON`.

Per esempio supponiamo di avere uno schermo normale a 16 colori (4 bit-planes) e di non voler considerare le collisioni degli sprite dispari. Se vogliamo rilevare una collisione tra uno sprite e il colore 13 dobbiamo scrivere nel registro CLXCON il valore

```
        111111
        5432109876543210
$03cb=%0000001111001101
```

Guardiamo il significato dei bit. I bit da 12 a 15 disabilitano gli sprite dispari. Dei bit da 6 a 11 sono a 1 solo i bit 6,7,8,9. Questo indica che solo i bit-planes da 1 a 4 sono abilitati per le collisioni. Si tratta dei soli bit-planes attivi. I bit da 0 a 5 contengono il numero %001101=13 cioè il numero del registro che ci interessa. Nel caso Dual Playfield la situazione è la stessa, solo che attivando tutti i bit-planes utilizzati per le collisioni si abilitano le collisioni con 2 colori contemporaneamente: per esempio, se con 2 playfield da 8 colori ciascuno vogliamo abilitare il rilevamento delle collisioni per il colore 7 del playfield 1 e il colore 2 del playfield 2 dobbiamo scrivere in `CLXCON` il numero

```
        111111
        5432109876543210
$0fbb=%0000111111011101
```

Questa combinazione di bit indica che tutti i bit-planes sono abilitati al rilevamento collisioni (tutti i bit da 6 a 11 valgono 1). Inoltre il numero del colore usato per il playfield 1 è dato dai bit 0,2 e 4 che messi affiancati formano il numero %111=7, mentre il numero del colore usato per il playfield 2 è dato dai bit 1,3 e 5 che messi affiancati formano il numero %010=2.

Da notare che la collisione di uno sprite con un colore del playfield 1 provoca l'accensione di un bit di `CLXDAT` diverso dal caso di collisione dello stesso sprite con un colore del playfield

2. Per esempio, come potete verificare nella tabella del registro CLXDAT, la collisione sprite 0 - playfield 1 mette a 1 il bit 1 di CLXDAT, mentre la collisione sprite 0 - playfield 2 mette a 1 il bit 5 di CLXDAT.

È possibile anche rilevare collisioni di uno sprite con più di un colore contemporaneamente sebbene solo in alcune particolari condizioni. Per capire come ciò sia possibile occorre tenere presente la rappresentazione binaria dei numeri dei registri colore. Ci sono, come sapete 32 registri colore numerati da 0 a 31. La possibilità di rilevare collisioni con 2 colori contemporaneamente si basa sul fatto che le rappresentazioni di alcuni numeri binari sono simili.

Per esempio consideriamo i numeri 2 e 21. In binario si ha 2=%00010 e 21=%10101 (consideriamo 5 bit per poter scrivere numeri fino a 31). Come vedete le rappresentazioni binarie di questi 2 numeri sono completamente diverse. Non vi è modo dunque di rilevare collisioni con entrambi i colori contemporaneamente.

Consideriamo invece i numeri 22 e 23. Osserviamo ora che espressi in binario 22=%010110 e 23=%010111. Le rappresentazioni dei 2 numeri differiscono solo per un bit, il bit più basso. In questo caso è possibile rilevare collisioni con entrambi i colori. Infatti il valore del bit più basso (che in questo caso differenzia i colori) è dato dal bit-plane 1. Se noi *non* abilitiamo il bit-plane 1 al rilevamento collisioni, verranno presi in considerazione solo i valori dei bit-plane 2,3,4 e 5 (siamo su uno schermo a 32 colori, quindi in totale 5 bit-planes), e il valore assunto dal bit-plane 1 non avrà alcuna influenza. Scriviamo dunque in CLXCON il valore:

```
         111111
         5432109876543210
CLXCON= %0000011110010110
```

Questo vuol dire che la collisione verrà rilevata basandosi sui soli bit-plane abilitati (e cioè 2,3,4 e 5) e precisamente quando il nostro sprite si sovrapporrà ad un pixel che abbia:

```
bitplane 1=(0 o 1) perché non è abilitato
bitplane 2=1
bitplane 3=1
bitplane 4=0
bitplane 5=1
```

Come abbiamo visto, sia la rappresentazione binaria di 22=%010010 che quella di 23=%010111 hanno questa particolare configurazione di bit, pertanto entrambi i colori provocano una collisione al passaggio dello sprite. Notate che il bitplane che NON abbiamo abilitato (l'1) corrisponde proprio all'unico bit che differenzia le rappresentazioni binarie di 22 e di 23.

Questa tecnica è applicabile a una qualunque coppia di colori le cui rappresentazioni binarie differiscano di un solo bit. Per esempio anche i numeri 8=%001000 e 9=%001001 differiscono per il bit più basso, quindi anche per rilevare collisioni tra lo sprite e questi 2 colori si deve disabilitare il bit-plane 1. Se invece consideriamo i colori 10=%001010 e 14=%001110, notiamo che le 2 rappresentazioni binarie differiscono nel bit 2 (numeriamo i bit da destra a sinistra a partire da 0) che corrisponde al bit-plane 3. Per rilevare collisioni tra lo sprite e questi 2 colori si deve disabilitare il bit-plane 3, e pertanto assegnare a CLXCON il valore riportato sotto:

```
         111111
         5432109876543210
CLXCON= %0000011011001010   ; bit 8=0 indica bit-plane 3 NON abilitato
```

Se disabilitiamo 2 bit-plane possiamo rilevare collisioni tra 4 colori. Il principio è sempre lo stesso. Prendiamo per esempio i colori:

```
1=%00001
3=%00011
5=%00101
7=%00111
```

questi 4 colori hanno i bit 0, 3 e 4 uguali tra loro, mentre si differenziano in quanto ogni colore ha una diversa combinazione di valori nei bit 1 e 2. Per rilevare collisioni tra uno sprite e tutti e 4 questi colori basta disabilitare i bit-plane 2 e 3 che corrispondono appunto ai bit 1 e 2. Disabilitando 3 bit plane si rilevano collisioni con 8 colori contemporaneamente, disabilitandone 4 con 16 colori.

Anche operando in modo Dual Playfield è possibile, per ciascun playfield, disabilitare alcuni bit-planes per rilevare collisioni tra lo sprite e più di un colore per ogni playfield (ricordiamo che se dobbiamo rilevare la collisione tra lo sprite e 2 colori che però appartengono uno al playfield 1 e uno al playfield 2 ciò non è necessario, perché in `CLXDAT` abbiamo per ogni playfield un bit che ci consente di rilevare contemporaneamente la collisione con entrambi i playfield).

In `lezione7x1.s` vediamo un esempio di collisione tra sprite e playfield in modo "standard". In `lezione7x2.s` invece c'è un esempio con il modo Dual Playfield. In entrambi i listati nel commento sono riportati diversi esempi di come rilevare collisioni con più di un colore per volta.

L'ultimo tipo di collisione è tra playfield 1 e playfield 2, ovviamente in modo Dual Playfield. È possibile rilevare una collisione tra uno o più colori del playfield 1 e uno o più colori del playfield 2, abilitando solo alcuni bit-planes esattamente con la stessa procedura adottata nel caso di collisione tra sprite e playfield. Quando viene rilevata una collisione tra i due playfield, bit 0 di `CLXDAT` assume il valore 1. Un esempio di questo tipo di collisione è in `lezione7x3.s`.

7.9 Uso diretto dei registri degli sprite

Vedremo ora un diverso metodo per utilizzare gli sprite. Finora abbiamo generato degli sprite utilizzando i registri `SPRxPT`, ovvero dei puntatori a delle strutture dati (dette strutture sprite) che contengono tutte le informazioni necessarie alla visualizzazione degli sprite. Esiste però un altro metodo per creare degli sprite che può essere usato in alternativa o anche in aggiunta a quello con i puntatori. Chiameremo questo nuovo metodo "uso diretto degli sprite". L'uso diretto degli sprite non è conveniente nella maggior parte dei casi, ma a volte può risultare utile. Per comprendere bene di cosa si tratti dobbiamo approfondire il discorso sulla visualizzazione degli sprite.

Quando memorizziamo in un registro `SPRxPT` l'indirizzo di una struttura sprite (secondo le modalità della tecnica "standard" di utilizzo degli sprite), attiviamo una procedura automatica che permette di visualizzare effettivamente gli sprite. Infatti i dati sulla posizione e la forma che noi abbiamo memorizzato nella struttura sprite, vengono trasferiti automaticamente, attraverso un "meccanismo" hardware chiamato *DMA*, in appositi registri, diversi dai registri `SPRxPT`; è proprio la scrittura dei dai in questi registri che REALMENTE permette la visualizzazione degli sprite. Del DMA, che è uno strumento molto importante dell'Amiga, diremo di più in una prossima lezione. Per il momento ci basta conoscere il ruolo che esso svolge nella visualizzazione degli sprite. Si comporta in pratica come un postino. Immaginate che la struttura dati dello sprite che voi avete costruito in memoria sia un mucchio di lettere indirizzate a diversi destinatari (registri). Il DMA si occupa di portare queste lettere a destinazione, smistandole tra i vari destinatari.

L'uso diretto degli sprite consiste, appunto, nello scrivere direttamente i dati degli sprite negli appositi registri, cioè nel portare "di persona" le lettere ai vari destinatari, rubando il lavoro al postino DMA. Visto che il DMA svolge il suo lavoro gratis, vi potreste chiedere a cosa serva questa tecnica. In effetti come abbiamo già detto di solito essa non offre vantaggi; tuttavia in alcuni casi può rivelarsi utile. Vediamo dunque in cosa consiste questa tecnica. Come abbiamo

già detto i dati degli sprite vengono scritti direttamente in alcuni registri. Ci sono 4 registri per ogni sprite, chiamati SPRxPOS, SPRxCTL, SPRxDATA, SPRxDATB (al posto della x dovete mettere il numero dello sprite che volete usare). Gli indirizzi di questi registri dipendono dallo sprite a cui ci si riferisce. Li possiamo calcolare con delle semplici formule. Con "x" indichiamo il numero dello sprite, da 0 a 7.

```
indirizzo SPRxPOS  = $dff140+(x*8)
indirizzo SPRxCTL  = $dff142+(x*8)
indirizzo SPRxDATA = $dff144+(x*8)
indirizzo SPRxDATB = $dff146+(x*8)
```

Potete comunque cercarli con l'help dell'ASMONE *<=C>*.

Ora descriviamo l'uso di questi registri. La forma di uno sprite viene scritta nei registri SPRxDATA e SPRxDATB, che costituiscono i 2 piccoli bit-planes dello sprite (SPRxDATB è il piano 2). Questi registri hanno lo stesso ruolo delle coppie di word che definiscono la forma di una riga dello sprite nella struttura sprite. Notate che per ogni sprite ci sono 2 registri che contengono i dati relativi ad *una sola* riga dello sprite. La posizione orizzontale di uno sprite come sapete è costituita da 9 bit, chiamati H0, H1 ...H8. Questi 9 bit sono suddivisi in due registri: il bit H0, cioè il bit basso si trova nel bit 0 del registro SPRxCTL. Gli altri 8 invece nel byte basso del registro SPRxPOS. In poche parole questi 2 registri si comportano, per quanto riguarda la posizione orizzontale, esattamente come le 2 word di controllo della struttura sprite. La posizione verticale, invece, con questa tecnica non viene determinata, perché gli sprite si comportano in modo piuttosto strano.

Per essere visualizzato, uno sprite deve essere attivato. Ciò accade quando si scrive nel registro SPRxDATA. Una volta attivato, lo sprite viene visualizzato ad ogni riga nella posizione orizzontale indicata, come abbiamo appena visto, nei registri SPRxPOS e SPRxCTL. La forma dello sprite è per ogni riga quella contenuta nei registri SPRxDATA e SPRxDATB. Quindi se il contenuto di questi registri non viene modificato ad ogni riga, lo sprite avrà la stessa forma ad ogni riga. Lo sprite viene visualizzato, fino a che non lo si disattiva scrivendo nel registro SPRxCTL. Per visualizzare uno sprite che cambi forma ad ogni riga si dovrebbe dunque utilizzare una copperlist fatta in questa maniera (supponiamo di usare lo sprite 0 e che sia VSTART=$40, VSTOP=$60, HSTART=$160):

```
1         dc.w    $4007,$fffe     ; WAIT - aspetta la linea VSTART
2         dc.w    $140,$0080      ; SPR0POS - posizione orizzontale
3         dc.w    $142,$0000      ; SPR0CTL
4         dc.w    $146,$0e70      ; SPR0DATB - forma sprite riga 1, piano 2
5         dc.w    $144,$03c0      ; SPR0DATA - forma sprite riga 1, piano 1
6                                 ; inoltre attiva la visualizzazione, per
7                                 ; questo va scritta per ultima.
8
9         dc.w    $4107,$fffe     ; WAIT - aspetta la linea VSTART+1
10        dc.w    $146,$0a70      ; SPR0DATB - forma sprite riga 2, piano 2
11        dc.w    $144,$0300      ; SPR0DATA - forma sprite riga 2, piano 1
12
13        dc.w    $4107,$fffe     ; WAIT - aspetta la linea VSTART+2
14        dc.w    $146,$0a7f      ; SPR0DATB - forma sprite riga 3, piano 2
15        dc.w    $144,$030f      ; SPR0DATA - forma sprite riga 3, piano 1
16
17  ; ripeti per ogni riga Y
18  ;       dc.w    $40+Y07,$fffe   ; WAIT - aspetta la linea VSTART+Y
19  ;       dc.w    $146,DATOY2     ; SPR0DATB - forma sprite riga Y, piano 2
20  ;       dc.w    $144,DATOY1     ; SPR0DATA - forma sprite riga Y, piano 1
21  ; mettendo al posto di DATOY1 e DATOY2 i dati della forma degli sprite.
22
23        dc.w    $6007,$fffe     ; WAIT - aspetta la linea VSTOP
24        dc.w    $142,$0000      ; SPR0CTL - disattiva lo sprite
```

Come vedete per sprite abbastanza alti è necessaria una copperlist molto lunga e complicata. In questo caso conviene decisamente usare il DMA. Supponiamo però di dover visualizzare uno

sprite che abbia la stessa forma ad ogni riga. Per esempio uno sprite che rappresenti una colonna. In questa situazione la nostra copperlist diventa semplicissima e cortissima (supponiamo di usare lo sprite 0 e che sia VSTART=$40, VSTOP=$60, HSTART=$160):

```
        dc.w    $4007,$fffe     ; WAIT - aspetta la linea VSTART
        dc.w    $140,$0080      ; SPR0POS - posizione orizzontale
        dc.w    $142,$0000      ; SPR0CTL
        dc.w    $146,$0e70      ; SPR0DATB - forma sprite riga 1, piano 2
        dc.w    $144,$03c0      ; SPR0DATA - forma sprite riga 1, piano 1
                                ; inoltre attiva la visualizzazione, per
                                ; questo va scritta per ultima.

        dc.w    $6007,$fffe     ; WAIT - aspetta la linea VSTOP
        dc.w    $142,$0000      ; SPR0CTL - disattiva lo sprite
```

Notate che la nostra copperlist, oltre a essere corta, non varia con l'altezza dello sprite. Al contrario, se volessimo usare il DMA per visualizzare questo sprite, saremmo costretti a memorizzare nella struttura dati le 2 word che rappresentano la forma tante volte quante sono le righe che costituiscono lo sprite. Pensate al caso in cui si deve visualizzare una colonna alta 100 righe. Se usassimo il DMA dovremmo memorizzare una struttura sprite che occupa molta memoria:

```
StrutturaSprite:
                dc.b    VSTART,HSTART,VSTOP,0
                dc.w    $ffff,$0ff0     ; riga 1
                dc.w    $ffff,$0ff0     ; riga 2
                dc.w    $ffff,$0ff0     ; riga 3
                dc.w    $ffff,$0ff0     ; riga 4
                dc.w    $ffff,$0ff0     ; riga 5
                dc.w    $ffff,$0ff0     ; riga 6
                dc.w    $ffff,$0ff0     ; riga 7
                dc.w    $ffff,$0ff0     ; riga 8

.... e cosi' via, fino a:

                dc.w    $ffff,$0ff0     ; riga 99
                dc.w    $ffff,$0ff0     ; riga 100
                dc.w    0,0             ; fine sprite
```

Con l'uso diretto degli sprite, invece basta una semplice copperlist:

```
        dc.b    VSTART,7,$ff,$fe        ; WAIT - aspetta la linea VSTART
        dc.w    $140
        dc.b    $00,HSTART      ; SPR0POS - posizione orizzontale
        dc.w    $142,$0000      ; SPR0CTL
        dc.w    $146,$ffff      ; SPR0DATB - forma sprite riga 1, piano 2
        dc.w    $144,$0ff0      ; SPR0DATA - forma sprite riga 1, piano 1
                                ; inoltre attiva la visualizzazione, per
                                ; questo va scritta per ultima.

        dc.b    VSTOP,7,$ff,$fe ; aspetta la linea VSTOP
        dc.w    $142,$0000      ; SPR0CTL - disattiva lo sprite
```

Un semplice esempio di uso diretto degli sprite è riportato in `lezione7y1.s`. Nel programma `lezione7y2.s`, invece, usando degli sprite in accesso diretto realizziamo delle barre verticali analoghe a quelle che si fanno orizzontalmente con il copper.

Con la tecnica dell'uso diretto degli sprite, è possibile anche visualizzare uno stesso sprite più volte su una stessa riga. Il metodo viene spiegato e applicato in `lezione7y3.s`. Gli sprite generati più volte sulla stessa linea sono anche detti *multiplexed*, cioè "multiplexati". Dunque eravamo partiti dicendo che ci sono 8 sprite solamente, ma abbiamo visto che l'assembler ci permette di moltiplicare gli sprite e anche di fargli assumere molti più colori di quelli standard, cambiando la palette più volte anche orizzontalmente. L'unico inconveniente è che ci vogliono delle copperlist molto lunghe, ma ne vale sicuramente la pena.

Uno sviluppo di questa idea ci porta a realizzare una schermata completamente grazie agli sprite, nell'esempio `Lezione7y4.s`.

Per compiere tale operazione però occorre scrivere una copperlist lunghissima, e per renderla più comprensibile sono stati usati dei *simboli* o *equates*, una direttiva del linguaggio assembler che permette di chiamare con un nome scelto a piacere un certo numero fisso, per cui scrivendo il nome viene assemblato il numero che gli corrisponde. Facciamo questo esempio: vogliamo fare in modo di accedere al registro COLOR0, che come sappiamo è $dff180. Possiamo scrivere:

```
1       move.w  #$123,$dff180
```

Ma se volessimo potremmo anche scrivere così:

```
1 COLORE0       EQU     $dff180         ; Definizione di un simbolo
2       move.w  #$123,COLORE0
```

In pratica abbiamo definito che quando l'asmone trova scritto COLORE0 deve assemblare come se avesse trovato $dff180. È come se definissimo una label, infatti bisogna inventarci un nome e scriverlo senza precederlo da spazi, ma non occorrono i : (in realtà si possono anche mettere i :, allo stesso modo le label potrebbero avere i : o non averle, l'ASMONE assembla comunque, ma certi assemblatori preferiscono che le LABEL siano seguite dai : e che i simboli (o equates) non li abbiano). EQU significa infatti EQUIVALE A. Quasi tutti gli assemblatori accettano anche il simbolo = al posto del simbolo EQU per la definizione. Facciamo un altro esempio:

```
1 NUMEROLOOP    =       10
2
3       MOVEQ   #NUMEROLOOP-1,d0
4 Loop:
5       clr.l   (a0)+
6       dbra    d0,NUMEROLOOP
7       rts
```

Con questo listatino azzeriamo 10 longword. L'utilità degli EQUATES è che possiamo metterli tutti all'inizio del listato, in modo che se vogliamo modificare certi valori, ad esempio quanti loop fare o quanti bitplanes puntare, basta modificare il valore del simbolo dopo l'= o l'EQU all'inizio del listato. Inoltre è possibile eseguire operazioni tra simboli. Un esempio pratico può essere il calcolo dello spazio da azzerare per un bitplane:

```
1  BytesPerRiga    =       40
2  NumeroRighe     =       256
3  SpazioBitplane  =       BytesPerRiga*NumeroRighe
4
5          ...
6
7          section plane,bss_C
8
9  Bitplane:
10         ds.b    SpazioBitplane
```

Nel listato SpazioBitplane vale 10240, ossia 40*256. In Lezione7y4.s vengono definiti dei simboli per la copperlist.

Infine nella lezione7y5.s faremo scorrere la schermata formata dagli sprite e ne approfitteremo per conoscere 2 nuove istruzioni del 68000, chiamate ROR e ROL. Le spiegheremo nel commento al listato.

7.10 Animazione sprite

Concludiamo questa lezione con una spiegazione sull'animazione degli sprite. Ritorniamo ora a considerare sprites "normali", cioè generati tramite i puntatori SPRxPT e il DMA. Per animare uno sprite è necessario cambiarne la forma ogni volta che esso viene ridisegnato. Ogni forma che viene assunta dallo sprite è detta "fotogramma dell'animazione". Di solito l'animazione è fatta in modo da avere una certa sequenza di fotogrammi che viene continuamente ripetuta. Pensate

per esempio ad un omino che cammina sullo schermo; noterete che tutti i passi sono uguali tra loro.

Per animare un omino che cammina sullo schermo si disegnano un certo numero di fotogrammi che visti in successione raffigurano un passo completo dell'omino. Quando l'omino ha completato il passo deve iniziarne uno nuovo: a questo punto si mostrano di nuovo gli stessi fotogrammi iniziando dal primo. Ripetendo per ogni passo sempre gli stessi fotogrammi possiamo mostrare l'omino camminare tutto il tempo che vogliamo, con un numero limitato di fotogrammi (è evidente che essendo i fotogrammi delle immagini, occupano memoria, quindi si deve cercare di usarne il meno possibile).

Fin qui il discorso è valido per un qualsiasi oggetto animato, e sarà bene che lo teniate presente anche quando tratteremo le animazioni realizzate tramite blitter. Ora invece ci occupiamo delle animazioni fatte tramite sprite. Questo vuol dire che abbiamo uno sprite che si muove sullo schermo e che ogni volta che viene ridisegnato assume una forma diversa. Di solito si procede in questo modo: per ogni fotogramma si realizza una struttura sprite, e ogni volta che lo sprite viene ridisegnato si fa puntare il registro `SPRxPT` ad un diverso fotogramma (ovvero ad una diversa struttura dati). La posizione dello sprite viene scritta ogni volta nella struttura del fotogramma a cui `SPRxPT` viene fatto puntare. Un esempio pratico è in `lezione7z.s`.

Questo esempio è anche il termine della lezione7 e del DISCO 1 del corso. Il disco 2 nel momento in cui scrivo (Maggio 1995) non è ancora del tutto terminato, comunque gli argomenti trattati sono:

- BLITTER (copymode, linemode e fill)
- Interrupt, CIAA/CIAB, caricamento da disco, Tastiera
- Audio
- Approfondimenti sul 68000, accenni al 68020
- Programmazione di videogiochi
- Routines matematiche (3d, frattali)
- Chipset AGA
- Compatibilità e ottimizzazioni
- Programmazione della scheda video PICASSO II!!!!

Non so se avrò tempo di terminare un lavoro così immane, già questo primo disco (in cui per la verità non ho trattato tanti argomenti) mi sembrava che fosse interminabile. Devo ringraziare Luca Forlizzi (The Dark Coder) per avermi aiutato a terminare la LEZIONE7, e anche i due volenterosi beta tester ANDREA SCARAFONI e FEDERICO STANGO che mi hanno indicato dove non ero stato chiaro, o addirittura dove avevo scritto frasi incomprensibili. Purtroppo non posso ringraziare coloro che hanno solo PROMESSO di aiutarmi, ma che poi sono spariti, come Alvise Spanò (AGA/LUSTRONES).

Per ricevere il disco 2, o almeno quello che avrò fatto in tempo a fare, potete scrivermi una lettera, o meglio sarebbe un disco con dei vostri programmini:

```
Fabio Ciucci
[omissis]
```

Non scrivetemi fino a che non avete veramente assimilato tutto quello che c'è in questo disco, non si diventa programmatori solamente avendo listati. Nel frattempo copiate a tutti questo disco, chiunque deve averlo, anche il Papa quando si affaccia dal balcone... chiunque abbia un Amiga in Italia deve avere questo disco. E non datelo solo a chi pensate che sia interessato, perché ho verificato che quelli che hanno seguito il corso e imparato di più sono quelli che meno mi aspettavo che lo facessi. Pensate che il più avanti qua a Lucca è un ragazzo, Michele, a cui avevo dato il disco perché lo copiasse ad un suo amico, che pareva fosse interessato. Non li ho più sentiti per un paio di mesi, poi mi si è presentato Michele a casa con la preview del suo gioco!!! Mentre il suo amico era rimasto impantanato! Quindi diffondete questo disco come fossero volantini per il vostro partito politico o opuscoli della vostra religione, scrivete annunci nelle bacheche o giornaletti delle vostre scuole o università per avvertire tutti che avete il disco per imparare a programmare l'Amiga, date delle copie del disco ai negozianti di Amiga della vostra zona chiedendogli di darli a chi chiede informazione sulla programmazione Amiga, insomma evangelizzate (assemblizzate) questo povero paese!

Quando poi sarà il momento di avere il disco 2, io spero di averlo finito, comunque vi manderò quello che c'è, dovreste scrivermi una lettera normale da 750 lire con una breve presentazione con età, segni particolari, computer posseduto, aspirazioni e quello che vi pare, con il vostro indirizzo, è ovvio. In concomitanza dovreste mandarmi un vaglia postale ad offerta libera, minimo 10.000 (per ora mi hanno mandato quasi tutti 10.000, qualcuno 20.000, poi sempre più rari i 30.000 e un mitico 60.000 da un Genovese, tanto per sfatare la leggenda che sono tirchi). Nella lettera dovete specificare quanto mi avete mandato di vaglia, perché lettera e vaglia di solito arrivano in tempi un pò diversi. Spero che non accadano ritardi galattici per le PT.

Altrimenti potete variare in questo modo: potete spedirmi un dischetto con la lettera in formato .txt, e magari qualche vostro listatino o altro per riempire il disco. Da notare che vi conviene mettere in disco in una busta da spedizione di quelle piccole, e di chiuderla solo con il fermacampioni o l'autoadesivo della busta stessa, senza mettere foglietti o lettere oltre al disco. Insomma non dovete megasigillare tutto con nastro adesivo o altro, questo per permettere l'ispezione postale, che tanto poi non la fanno mai, ma se il pacco è così si paga meno.

Dovete poi scrivere "pacchetto" sulla busta, e dire alla (spero) gentile impiegata dell'ufficio postale che si tratta di un pacchetto senza lettere dentro, e questo vi farà risparmiare rispetto ad un pacco chiuso con lettera, che conta come "LETTERA PESANTE"! Un pacchetto così con 1 solo dischetto dovrebbe costare 1200 lire. Se volete che arrivi prima, potete farlo espresso, aggiungendo 3000 lire, ma allora vi conviene calcolare il tempo che vi manca a finire li disco 1 e spedire per tempo... e che programmatori siete se non siete furbi? NOTA: Se volete che io vi mandi il disco 2 in espresso, potete fare i calcoli come prima, oppure partire da una base minima di 13000 lire e indicare nella lettera (cartacea o .txt) che volete il disco 2 espresso.

Un ultimo sistema potrebbe essere quello di mettere i soldi "sciolti" nella lettera o nel pacchetto, ma occorre metterli in modo che non si vedano in trasparenza, e il pacchetto allora dovrebbe essere fatto chiuso... decidete voi la strada, ma credo che lettera o pacchetto + vaglia sia meglio.

Eventuali donazioni o contatti incoraggeranno me e i miei "collaboratori" a continuare l'opera di stesura di questo corso. Saranno graditissime anche eventuali offerte di collaborazione da parte di programmatori esperti, in particolare di coloro che hanno programmato giochi (quelli che fanno le demo li conosco tutti qua in Italia, la scena è una specie di famiglia, con i litigi delle famiglie, però!).

Devo anche segnalarvi il miglior club di utenti Amiga, dove potreste trovare altri programmatori, grafici o musicisti per collaborazioni:

```
Amiga Expert Team
```

```
Mirko Lalli
[omissis]
```

CAPITOLO 8

LEZIONE 8 - APPROFONDIMENTI SUL 68000

In questa lezione verranno approfondite le conoscenze sul 68000 e saranno fatte delle precisazioni su vari argomenti già trattati. Una nota per chi installa il corso su HardDisk: vi consiglio di fare delle directory, con il nome del relativo disco del corso:

```
Assembler1
Assembler2
Assembler3
...
```

Dove copiare gli interi dischetti. Poi aggiungete ad `s:startup-sequence`:

```
assign Assembler1: dh0:Assembler1
assign Assembler2: dh0:Assembler2
assign Assembler3: dh0:Assembler3
...
```

(`dh0:` è solo un esempio... ci metterete il drive opportuno, naturalmente!). Poi vi consiglierei di scompattare tutti i sorgenti e i dati, che sono in formato powerpacker. Per fare ciò, copiate il file `c:PP` nell'HardDisk, e verificate se avete in `LIBS:` la `PowerPacker.library`, altrimenti copiatela da questo disco. Ora, eseguite dallo shell `PP`, in modo da abilitare la scompattazione automatica. Ora fatevi una directory "provvisoria", per esempio chiamatela "buffer". Se copiate TUTTI i file dalla directory Assembler1 alla directory Buffer, tutti i file saranno scompattati, infatti si "allungheranno". Ora potete ricopiarli tutti in Assembler1 (magari con un `MOVE` del DiskMaster o di DirOpus, che li ricancella anche da buffer). Allo stesso modo potete ricopiare tutto Assembler2 in buffer, poi ricopiare in Assembler2. Per risparmiare questa pur veloce operazione, potete caricare il PP prima di copiare i file da dischetto alle directory su HD, in modo che i file in AssemblerX siano scompattati. I 3 comandi assign servono per fare in modo che anziché cercare il disco con il nome "AssemblerX:" si cerchi nella directory `dh0:AssemblerX`. In alcuni dei prossimi listati, infatti, si cerca `Assembler2:`, e così sarà anche per `Assembler3:`.

P.S: Ho intenzione di tradurre in Inglese l'intero corso. Però questo mi costringerebbe a non scrivere più nuove lezioni per MESI interi... Quindi, se trovassi qualcuno che abbia già letto il disco 1, che sappia l'inglese decentemente, e abbia voglia di tradurre almeno una lezione, io

sarei felicissimo. Chi mi aiutasse nell'opera di traduzione naturalmente avrebbe una percentuale molto alta sui profitti ricavati dall'estero (che ne dite del 30%? Forse è troppo...) Chi può aiutarmi, (sto parlando di un BEL lavoro di traduzione), mi contatti quanto prima.

P.S2: Mi raccomando di copiare a TUTTI i vostri amici (e non) il disco 1 del corso, di darlo ai negozianti della vostra città, di mettere annunci nelle bacheche o nei giornaletti per vedere se interessa a qualcuno, e trovare nuovi contatti per programmare. In particolare potreste diffondere la filosofia CyberAssemblica della scena, di cui trovate un sunto nel file `SCENA.TXT`. Anche il Papa quando si affaccia dal balcone deve avere il disco 1 del corso! (liberamente copiabile). Per quanto riguarda il disco 2, ossia questo, invece, non è liberamente copiabile, altrimenti io poi non prenderei nemmeno quei (non molti) soldi che mi mandano coloro che hanno avuto solo il disco 1. Immaginiamoci se costoro avessero subito entrambi i dischi! Comunque, quando (e se) farò dischi 3, 4, eccetera, probabilmente renderò anche il disco 2 liberamente copiabile (shareware, però), in modo che i nuovi possano avere subito i disk1+2, poi io mi rifaccio qualche spicciolo col disco 3, 4...

Continuiamo la lezione, sia che il file sia su HardDisk che su Floppy. Innanzitutto è necessario completare la lezione sul 68000, dato che per ora ne è stato fatto un uso semplificato. Già dalla lezione precedente avete potuto constatare che spessissimo è necessario operare sui singoli bit dei numeri o dei registri, ebbene più andrete avanti nella programmazione e più tenderete ad inserire istruzioni come `AND`, `OR`, `NOT`, `ROL`, `ASL`...eccetera, ossia le operazioni logiche booleane e di scorrimento sui bit. Ah! Nella directory LEZIONI è presente un testo che spiega cosa è la SCENA AMIGA. Ora che state diventando dei coder è opportuno che sappiate chi dovete ringraziare per la nascita della cultura della programmazione delle demo, in quel modo "illegale", che come avete visto però funziona, e anche molto bene. Il testo è `SCENA.TXT`, leggetevelo quando non avete voglia di farvi fumare il cervello con le lezioni di asm!

Prima di procedere nel corso, è necessario fare un listato di startup, ossia di salvataggio e ripristino della copper di sistema, più efficiente di quello usato fino ad adesso, inoltre tale startup dovrà essere inclusa in tutti i prossimi listati, dunque sarà certamente più utile caricarla tramite la direttiva `INCLUDE` già vista per includere la routine che suona la musica. "Costruiremo" questa startup nella lezione passo dopo passo, come risultato delle varie precisazioni. Analizziamo la procedura di startup usata nelle lezioni precedenti:

```
1  Inizio:
2          move.l  4.w,a6              ; Execbase
3          jsr     -$78(a6)            ; Disable
4          lea     GfxName(PC),a1      ; Nome lib
5          jsr     -$198(a6)           ; OpenLibrary
6          move.l  d0,GfxBase
7          move.l  d0,a6
8          move.l  $26(a6),OldCop      ; salviamo la vecchia COP
9
10 ; Qua è puntata la nostra copperst e ci sono le routines
11
12         move.l  OldCop(PC),$dff080      ; Puntiamo la cop di sistema
13         move.w  d0,$dff088              ; facciamo partire la cop
14         move.l  4.w,a6
15         jsr     -$7e(a6)            ; Enable
16         move.l  gfxbase(PC),a1
17         jsr     -$19e(a6)           ; Closelibrary
18         rts
```

In pratica arrestiamo il multitasking e gli interrupt di sistema tramite il Disable, poi apriamo la libreria grafica, tramite la quale possiamo trovare l'indirizzo della vecchia copperlist, sapendo che si trova $26 bytes dopo l'indirizzo di *GfxBase*. Sapendo come rimettere a posto la vecchia copperlist e avendo immobilizzato il WorkBench, agiamo in modo diretto sui chip custom senza temere incompatibilità. Al termine delle routines sarà necessario eseguire l'Enable per riattivare il multitasking e puntare la vecchia copperlist per rivisualizzare le finestrelle del sistema

operativo. Queste operazioni sono il minimo indispensabile per poter lavorare, ma si potrebbero fare dei perfezionamenti al codice, per esempio si potrebbero eseguire delle routines della libreria grafica che resettano il modo video, in modo da resettare anche modi video per monitor VGA/Multisync/Multiscan o altri. Una funzione apposita esiste, e si chiama `LoadView`, vediamola:

```
; Abbiamo il GfxBase nel registro A6

        MOVE.L  $22(A6),WBVIEW  ; Salva il WBView attuale di sistema
        SUBA.L  A1,A1           ; View nullo per azzerare il modo video
        JSR     -$DE(A6)        ; LoadView nullo - modo video azzerato
```

La funzione `LoadView` richiede che sia specificato l'indirizzo della struttura view in `a1`, ma in questo caso `A1` è AZZERATO, dato che sommiamo `a1` a se stesso, ricavando `a1=0`. Quando `A1` è NULLO la funzione resetta il modo video riportandolo ad un LOWRES non interlacciato e senza frequenze speciali per i monitor. A questo punto siamo più sicuri di avere la situazione della copperlist sotto controllo, inoltre abbiamo salvato il vecchio puntatore alla struttura `WBVIEW` in una label, che ci permetterà alla fine del listato di ripristinarlo, e con esso le eventuali frequenze speciali per monitor:

```
        MOVE.L  WBVIEW(PC),A1   ; Vecchio WBVIEW in A1
        MOVE.L  GFXBASE(PC),A6  ; GFXBASE in A6
        JSR     -$DE(A6)        ; loadview - rimetti il vecchio View
```

Per essere sicuri che anche il modo interlacciato sia resettato e ripristinato correttamente, si può attendere per due fotogrammi tramite l'esecuzione della routine asmWaitOF, sempre della `graphics.library`:

```
        MOVE.L  WBVIEW(PC),A1   ; Vecchio WBVIEW in A1
        MOVE.L  GFXBASE(PC),A6  ; GFXBASE in A6
        JSR     -$DE(A6)        ; loadview - rimetti il vecchio View
        JSR     -$10E(A6)       ; WaitOf ( Risistema l'eventuale interlace)
        JSR     -$10E(A6)       ; WaitOf
```

Per avere l'animo in pace, mettiamo un paio di `WaitOF` anche dopo il primo `loadwiew` che resetta il modo video, e già che ci siamo controlliamo se il reset è veramente avvenuto testando se il `WBVIEW` è azzerato come previsto:

```
; Abbiamo il GfxBase nel registro A6

        MOVE.L  $22(A6),WBVIEW  ; Salva il WBView attuale di sistema
        SUBA.L  A1,A1           ; View nullo per azzerare il modo video
        JSR     -$DE(A6)        ; LoadView nullo - modo video azzerato
        JSR     -$10E(A6)       ; WaitOf ( Queste due chiamate a WaitOf    )
        JSR     -$10E(A6)       ; WaitOf ( servono a resettare l'interlace )
```

Avendo usato routines del sistema operativo, siamo certi che in macchine del futuro il modo video sarà comunque azzerato. Per "esagerare" nella compatibilità, possiamo richiamare alla fine del listato delle funzioni della `intuition.library` che "ridisegnano" gli schermi e le finestrelle:

```
        move.l  4.w,a6          ; ExecBase in A6
        LEA     IntuiName(PC),A1 ; Nome libreria da aprire (intuition)
        JSR     -$198(A6)       ; OldOpenLibrary - apri la lib
        TST.L   D0              ; Errore?
        BEQ.s   EXIT            ; Se si, esci senza eseguire il codice
        MOVE.L  D0,A6           ; IntuiBase in a6
        jsr     -$186(A6)       ; ReThinkDisplay - Riordina i connotati degli
                                ; schermi...
```

Questa operazione è analoga a quella svolta col `WBView`. Per ora non abbiamo ancora usato il blitter, ma nelle prossime lezioni ci saranno molte blittate, e siccome useremo questa startup, sarà utile predisporla per tale scopo. Basta accertarsi che il blitter non sia usato dal sistema operativo mentre lo stiamo usando noi, ed esiste una funzione della `GfxLib` in grado di far cessare l'utilizzo del blitter da parte del WorkBench:

```
1         jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
2                                 ; impedendone l'uso al sistema operativo.
```

Al termine del listato, basterà chiamare la funzione che fa l'opposto, ossia riabilita l'uso del blitter da parte della `graphics.library`:

```
1         jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
2                                 ; può nuovamente usare il blitter
```

Queste due funzioni sono simili al Disable e all'Enable, che come abbiamo visto fermano il multitasking e gli interrupts di sistema e li riabilitano. In realtà esiste anche una funzione meno drastica del Disable, cioè il Forbid, il quale disabilita il multitasking lasciando in funzione gli interrupt di sistema; nessuno vieta di usare insieme Forbid e Disable, forse rende l'arresto del sistema meno brusco, proviamoli insieme:

```
1         move.l  4.w,a6          ; ExecBase in A6
2         JSR     -$84(a6)        ; FORBID - Disabilita il Multitasking
3         JSR     -$78(A6)        ; DISABLE - Disabilita anche gli interrupt
4                                 ;           del sistema operativo
5 ; routines
6
7         MOVE.L  4.w,A6          ; ExecBase in a6
8         JSR     -$7E(A6)        ; ENABLE - Abilita System Interrupts
9         JSR     -$8A(A6)        ; PERMIT - Abilita il multitasking
```

Ora l'Amiga non può protestare con un Guru Meditation o un SoftWare Failure dicendo che non l'avevamo avvertita che stavamo programmando l'hardware!

Dato che salviamo lo status di tutto, perché non salvare i valori dei registri dati ed indirizzi? Esiste una istruzione che viene usata prevalentemente per questo scopo, ed è il `MOVEM`. I registri però sono salvati nello STACK, ossia il registro `A7`, detto anche `SR`, che per ora abbiamo evitato di usare. Vediamo cosa è lo stack: pensate che si tratta di un registro analogo ad un registro indirizzi, non per niente è il registro `A7`, dunque il valore che contiene è un indirizzo, ossia PUNTA ad un indirizzo. Il fatto è che se modifichiamo l'indirizzo contenuto in `A7` (o `SP`) l'Amiga impazzisce totalmente. Ma chi mette quell'indirizzo nello Stack Pointer? Dato che modificandolo si verifica il Guru/Software Failure, potrete intuire che è il sistema operativo che decide tale numero ogni reset, ed è lui a modificarlo quando serve. Sapendo come usarlo, però può esserci molto utile. Abbiamo visto nel corso come sia possibile indicare una zona di memoria con l'indirizzamento indiretto, ad esempio scrivendo:

```
1         lea     bitplane,a0
2         move.l  #$123,(a0)+
3         move.l  #$456,(a0)+
```

Abbiamo inserito i valori $123 e $456 nel bitplane agendo sul registro `a0`, dato che abbiamo fatto PUNTARE `a0` al bitplane. Da questo listatino vediamo anche come sia possibile, con l'indirizzamento indiretto con post-incremento, inserire dati consecutivamente, uno dopo l'altro, nella zona di memoria. Cosa succederebbe se, dopo quelle istruzioni, scrivessimo:

```
1         move.l  -(a0),d0
2         move.l  -(a0),d1
```

Succederebbe che in `d0` sarebbe copiato l'ultimo valore inserito, ossia $456, mentre in `d1` il primo, $123, e `a0` punterebbe di nuovo a bitplane. In pratica siamo "tornati indietro". Ebbene, immaginate di fare l'operazione opposta a questa: nel caso che abbiamo visto, c'è una zona di memoria, che abbiamo chiamato BITPLANE, e scriviamo da quell'indirizzo in avanti con i `move.l #xxx,(a0)+`

```
Bitplane
   o------------>
```

Poi, dopo un certi numero di istruzioni, a0 punta a bitplane+x, cioè molto più avanti in memoria. Possiamo "riprendere" i valori che abbiamo "seminato" in questo campo con dei move.l -(a0),xxx che ci fanno tornare indietro fino a raggiungere nuovamente l'indirizzo di partenza BITPLANE. Ma attenzione! abbiamo raccolto i dati nell'ordine inverso rispetto a quello di immissione, infatti l'ultimo inserito è il primo ripreso. Lo stack punta ad un indirizzo in memoria, che serve da "campo" in cui seminare, ossia da zona dove salvare e riprendere dati. Bisogna stare attenti, però, che viene usato "all'indietro", al contrario dell'esempio del bitplane. L'esigenza dello stack nasce con i primi CPU, ed è organizzato in questo modo: la memoria di un computer di solito viene riempita dalle locazioni più basse fino alle più alte, per esempio se abbiamo un computer con 512k di memoria e dobbiamo caricarci un file lungo 256k, saranno riempiti i primi 256k e rimarranno liberi i Kb dal 257 al 512. Volendo riservare uno spazio STACK per salvare dei dati generici, si è pensato di far partire questo spazio dalla fine della memoria circa, e di salvare i dati "all'indietro" verso la prima locazione di memoria, in modo da utilizzare al meglio la memoria:

```
ZERO ---------------------------------------FINE MEMORIA
     Programmi ----->>          <<-----STACK
```

In questo modo lo stack non viene sovrascritto a meno che la memoria non sia proprio finita, e comunque i programmi sotto sistema operativo evitano tale scontro! Noi dobbiamo fare demo o giochi eseguibili dal sistema operativo, dunque dobbiamo usare lo stack in modo standard per non creare conflitti o sovrascritture. Se facessimo un programma in autoboot e senza bisogno di uscita, potremmo definire un'area nostra per lo stack, ma questo può generare problemi di compatibilità e per ora vi consiglio di non farlo. Vediamo infine come immettere e prelevare dati dallo STACK, cominciando con un esempio semplicissimo: salvare il contenuto del registro D0, e successivamente ripristinarlo.

```
1        MOVE.L  d0,-(SP)        ; salviamo d0 nello stack. NOTA: se dobbiamo
2                                ; salvare un solo registro, usiamo MOVE e
3                                ; non MOVEM, usato per Multipli registri.
4
5  ;     Routines che modificano D0
6
7        MOVE.L  (SP)+,d0        ; ripristiniamo il vecchio valore di d0
8                                ; prendendolo dallo STACK
```

Da notare che scrivere MOVE.L d0,-(SP) o MOVE.L d0,-(A7) è equivalente, in memoria viene assemblata la stessa sequenza binaria. Notiamo che il contenuto di d0 viene copiato nell'indirizzo cui punta SP, e SP stesso viene a puntare una long più indietro. Poi d0 viene modificato da varie routines, e quando vogliamo ottenere il suo vecchio valore basta riprenderlo dall'indirizzo in SP, e notate bene che con (SP)+ riportiamo SP a puntare l'indirizzo che puntava prima di aver salvato d0, cioè siamo andati indietro di una long, poi siamo ritornati avanti ripescando il valore. Proviamo ora a salvare il valore di più registri:

```
1        MOVE.L  d0,-(SP)        ; salviamo d0 nello stack
2        MOVE.L  d1,-(SP)        ; salviamo d1 nello stack
3        MOVE.L  d2,-(SP)        ; salviamo d2 nello stack
4        MOVE.L  d3,-(SP)        ; salviamo d3 nello stack
5
6  ;     Routines che modificano d0,d1,d2,d3
7
8        MOVE.L  (SP)+,d3        ; ripristiniamo il vecchio valore di d3
9        MOVE.L  (SP)+,d2        ; ripristiniamo il vecchio valore di d2
10       MOVE.L  (SP)+,d1        ; ripristiniamo il vecchio valore di d1
11       MOVE.L  (SP)+,d0        ; ripristiniamo il vecchio valore di d0
12                               ; prendendoli dallo STACK
```

Notate che l'ultimo valore salvato è il primo che può essere ripescato, proprio per il fatto che andiamo indietro e poi ritorniamo avanti, leggendo dall'ultimo valore immesso al contrario fino al primo:

```
                                        Indirizzo di partenza STACK
IMMISSIONE:     -(SP)   <--------------o        - indietro -


                                        Indirizzo di partenza STACK
LETTURA:        (SP)+   --------->     o        - avanti -
```

è una struttura detta a “pila”, infatti può essere immaginata in questo modo: pensate di avere una collezione di fumetti, e di volerla ordinare dal numero uno al numero 50. Trovato il numero 1, lo mettete su un tavolo. Trovato il numero 2 lo mettete sopra il numero 1. Poi il 3 sul 2, e via via fate una “pila” di fumetti, fino a che non avrete messo il numero 50 in cima alla catasta. Ora, se voleste riprendere i fumetti, il primi che si presenta è il 50, poi sotto trovate il 49, il 48 eccetera, e come ultimo trovate l’1. Lo stack infatti è del tipo “first in, last out”, ossia “il primo messo dentro è l’ultimo tirato fuori”. Capirete che modificando impropriamente lo stack vengono presi valori in memoria a caso e considerati come valori salvati prima. Dunque state attenti, quando avete eseguito un:

```
1       MOVE.L  xxxx,-(SP)      ; salviamo xxxx nello stack
```

La prossima volta che leggete con `(SP)+` dallo stack ricavate `xxxx`.

Nello stack potete salvare e riprendere qualsiasi dato, ma una delle più evidenti utilità è quella di salvare lo stato dei registri, e per fare questo si può usare il semplice `MOVE.L`, nel caso già visto del salvataggio di un solo registro, oppure il `MOVEM` (MOVE Multiple) per più registri. Vediamo come funziona il `MOVEM`: per salvare tutti i registri (escluso `a7`, ovviamente, che è l’`SP`, dunque `d0,d1,d2,d3,d4,5,d6,d7,a0,a1,a2,a3,a4,a5,a6`), si deve eseguire questo solo `MOVEM` anziché 15 `MOVE`:

```
1       MOVEM.L d0-d7/a0-a6,-(SP)       ; salva tutti i registri nello STACK
```

E per ripristinarli tutti basta un:

```
1       MOVEM.L (SP)+,d0-d7/a0-a6       ; riprendi tutti i registri dallo STACK
```

Praticamente il `MOVEM` sposta una lista di registri nella destinazione, nel caso del `MOVEM.L d0-d7/a0-a6,destinazione`, oppure copia una sorgente in vari registri, nel caso di `MOVEM.L sorgente,d0-d7/a0-a6`. La sorgente e la destinazione sono in formato “standard”, per cui si può copiare da e verso LABEL/INDIRIZZI o indirizzamenti indiretti:

```
1       MOVEM.L d0-d7/a0-a6,-(SP)
2       MOVEM.L d0-d7/a0-a6,LABEL
3       MOVEM.L d0-d7/a0-a6,$53000
4
5       MOVEM.L $53000,d0-d7/a0-a6
6       MOVEM.L LABEL(PC),d0-d7/a0-a6
7       MOVEM.L (SP)+,d0-d7/a0-a6
```

La lista segue questo standard: si possono indicare i registri separatamente, separandoli con la barretta “/”, per cui si può dire che:

```
1       MOVEM.L d0-d7/a0-a6,-(SP)
```

è equivalente a:

```
1       MOVEM.L d0/d1/d2/d3/d4/d5/d6/d7/a0/a1/a2/a3/a4/a5/a6,-(SP)
```

Ma, poiché le serie consecutive di registri possono essere indicati ponendo il primo registro della serie e l’ultimo separati da un “-”, si dividono con la barretta solamente i registri dati da quelli indirizzi. In realtà l’asmone accetta anche:

```
1       MOVEM.L d0-a6,-(SP)
```

Considerandolo come la lunghissima istruzione precedente, ma dato che non tutti gli assemblatori accettano questa forma, è meglio mettere la "/" tra le serie di registri dati e quella dei registri indirizzi. Facciamo degli esempi: vogliamo salvare i registri `d0,d1,d2,d5 e a3`:

```
1       MOVEM.L d0-d2/d5/a3,-(SP)
```

Abbiamo semplificato `d0/d1/d2` con `d0-d2`. Proviamo ora a salvare `d2,d4,d5,d6,a3,a4,a5,a6`:

```
1       MOVEM.L d2/d4-d6/a3-a6,-(SP)
```

Chiaramente per ripristinare questi registri si scriverà:

```
1       MOVEM.L (SP)+,d2/d4-d6/a3-a6
```

Credo che la sintassi del `MOVEM` sia chiara. Tramite questa istruzione è possibile gestire il multitasking, infatti vi siete mai chiesti come è possibile far girare due programmi insieme, che usano gli stessi registri dati e indirizzi, senza che interferiscano fra loro: la risposta è semplice! All'inizio di ogni routine c'è un `MOVEM` che salva lo stato dei registri, la routine viene eseguita, e all'uscita i registri tornano al loro stato originario come se tale routine non fosse mai stata eseguita. Molte routines infatti sono così strutturate:

```
1   Routine:
2       MOVEM.L d0-d7/a0-a6,-(SP)
3       ....
4       ....
5       MOVEM.L (SP)+,d0-d7/a0-a6
6       rts
```

In questo modo, un `BSR.W ROUTINE` non causa la modifica dei registri, per cui se in `a5` c'era *$dff000* e in `a6` *ExecBase*, siamo sicuri che dopo aver eseguito la routine ci sono sempre questi valori. Nell'usare il `MOVEM` capita spesso, le prime volte, di perdere "il filo" dei `movem` già fatti per cui può succedere una cosa del genere:

```
1   Routine:
2       MOVEM.L d0-d7/a0-a6,-(SP)
3       ....
4       ....
5       MOVEM.L (SP)+,d0-d7/a0-a6
6       ....
7       ....
8       MOVEM.L (SP)+,d0-d7/a0-a6
9       rts
```

In questo caso c'è un **errore madornale**, perché come prima cosa lo stack è andato troppo avanti, per cui tutti i dati che verranno ripresi dallo stack in seguito saranno sbagliati, come seconda cosa già i registri avranno valori diversi da quelli in entrata. Per far tornare tutto si potrebbe fare così:

```
1   Routine:
2       MOVEM.L d0-d7/a0-a6,-(SP)
3       ....
4       ....
5       ....
6       MOVEM.L d0-d7/a0-a6,-(SP)
7       ....
8       ....
9       MOVEM.L (SP)+,d0-d7/a0-a6
10      ....
11      ....
12      MOVEM.L (SP)+,d0-d7/a0-a6
13      rts
```

In questo modo all'uscita della routine i registri hanno il valore di entrata e lo stack è tornato all'indirizzo di entrata. (entrata nella routine!)

A questo punto possiamo dotare la nostra startup di salvataggio iniziale dei registri e ripristino finale, analogamente a questo ultimo esempio. Ecco come si presenta la nostra startup in questo momento:

```
1  MAINCODE:
2          movem.l d0-d7/a0-a6,-(SP)       ; Salva i registri nello stack
3          move.l  4.w,a6                  ; ExecBase in a6
4          LEA     GfxName(PC),A1          ; Nome libreria da aprire
5          JSR     -$198(A6)               ; OldOpenLibrary - apri la lib
6          MOVE.L  d0,GFXBASE              ; Salva il GfxBase in una label
7          BEQ.w   EXIT2                   ; Se si, esci senza eseguire il codice
8          LEA     IntuiName(PC),A1        ; Intuition.lib
9          JSR     -$198(A6)               ; Openlib
10         MOVE.L  D0,IntuiBase
11         BEQ.w   EXIT1                   ; Se zero, esci! Errore!
12
13         MOVE.L  IntuiBase(PC),A0
14         CMP.W   #39,$14(A0)     ; versione 39 o maggiore? (kick3.0+)
15         BLT.s   VecchiaIntui
16         BSR.w   ResettaSpritesV39
17 VecchiaIntui:
18
19         MOVE.L  GfxBase(PC),A6
20         MOVE.L  $22(A6),WBVIEW  ; Salva il WBView attuale di sistema
21
22         SUBA.L  A1,A1           ; View nullo per azzerare il modo video
23         JSR     -$DE(A6)        ; LoadView nullo - modo video azzerato
24         SUBA.L  A1,A1           ; View nullo
25         JSR     -$DE(A6)        ; LoadView (due volte per sicurezza...)
26         JSR     -$10E(A6)       ; WaitOf ( Queste due chiamate a WaitOf    )
27         JSR     -$10E(A6)       ; WaitOf ( servono a resettare l'interlace )
28         JSR     -$10E(A6)       ; Altre due, vah!
29         JSR     -$10E(A6)
30
31         MOVEA.L 4.w,A6
32         SUBA.L  A1,A1           ; NULL task - trova questo task
33         JSR     -$126(A6)       ; findtask (d0=task, FindTask(name) in a1)
34         MOVEA.L D0,A1           ; Task in a1
35         MOVEQ   #127,D0         ; Priorità in d0 (-128, +127) - MASSIMA!
36         JSR     -$12C(A6)       ; _LVOSetTaskPri (d0=priorità, a1=task)
37
38         MOVE.L  GfxBase(PC),A6
39         jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
40                                 ; impedendone l'uso al sistema operativo.
41         jsr     -$E4(A6)        ; WaitBlit - Attende la fine di ogni blittata
42         JSR     -$E4(A6)        ; WaitBlit
43
44         move.l  4.w,a6          ; ExecBase in A6
45         JSR     -$84(a6)        ; FORBID - Disabilita il Multitasking
46         JSR     -$78(A6)        ; DISABLE - Disabilita anche gli interrupt
47                                 ;           del sistema operativo
48 **************
49         bsr.w   HEAVYINIT       ; Ora puoi eseguire la parte che opera
50 **************                 ; sui registri hardware
51
52         move.l  4.w,a6          ; ExecBase in A6
53         JSR     -$7E(A6)        ; ENABLE - Abilita System Interrupts
54         JSR     -$8A(A6)        ; PERMIT - Abilita il multitasking
55
56         SUBA.L  A1,A1           ; NULL task - trova questo task
57         JSR     -$126(A6)       ; findtask (d0=task, FindTask(name) in a1)
58         MOVEA.L D0,A1           ; Task in a1
59         MOVEQ   #0,D0           ; Priorità in d0 (-128, +127) - NORMALE
60         JSR     -$12C(A6)       ; _LVOSetTaskPri (d0=priorità, a1=task)
61
62         MOVE.W  #$8040,$DFF096  ; abilita blit
63         BTST.b  #6,$dff002      ; WaitBlit...
64 Wblittez:
65         BTST.b  #6,$dff002
66         BNE.S   Wblittez
67
68         MOVE.L  GFXBASE(PC),A6  ; GFXBASE in A6
69         jsr     -$E4(A6)        ; Aspetta la fine di eventuali blittate
70         JSR     -$E4(A6)        ; WaitBlit
71         jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
72                                 ; può nuovamente usare il blitter
73         MOVE.L  IntuiBase(PC),A0
```

```
74          CMP.W   #39,$14(A0)      ; V39+?
75          BLT.s   Vecchissima
76          BSR.w   RimettiSprites
77  Vecchissima:
78
79          MOVE.L  GFXBASE(PC),A6   ; GFXBASE in A6
80          MOVE.L  $26(a6),$dff080  ; COP1LC - Punta la vecchia copper1 di sistema
81          MOVE.L  $32(a6),$dff084  ; COP2LC - Punta la vecchia copper2 di sistema
82          JSR     -$10E(A6)        ; WaitOf ( Risistema l'eventuale interlace)
83          JSR     -$10E(A6)        ; WaitOf
84          MOVE.L  WBVIEW(PC),A1    ; Vecchio WBVIEW in A1
85          JSR     -$DE(A6)         ; loadview - rimetti il vecchio View
86          JSR     -$10E(A6)        ; WaitOf ( Risistema l'eventuale interlace)
87          JSR     -$10E(A6)        ; WaitOf
88          MOVE.W  #$11,$DFF10C     ; Questo non lo ripristina da solo..!
89          MOVE.L  $26(a6),$dff080  ; COP1LC - Punta la vecchia copper1 di sistema
90          MOVE.L  $32(a6),$dff084  ; COP2LC - Punta la vecchia copper2 di sistema
91          moveq   #100,d7
92  RipuntLoop:
93          MOVE.L  $26(a6),$dff080  ; COP1LC - Punta la vecchia copper1 di sistema
94          move.w  d0,$dff088
95          dbra    d7,RipuntLoop    ; Per sicurezza...
96
97          MOVEA.L IntuiBase(PC),A6
98          JSR     -$186(A6)        ; _LVORethinkDisplay - Ridisegna tutto il
99                                   ; display, comprese ViewPorts e eventuali
100                                  ; modi interlace o multisync.
101         MOVE.L  a6,A1            ; IntuiBase in a1 per chiudere la libreria
102         move.l  4.w,a6           ; ExecBase in A6
103         jsr     -$19E(a6)        ; CloseLibrary - intuition.library CHIUSA
104 Exit1:
105         MOVE.L  GfxBase(PC),A1   ; GfxBase in a1 per chiudere la libreria
106         jsr     -$19E(a6)        ; CloseLibrary - graphics.library CHIUSA
107 Exit2:
108         movem.l (SP)+,d0-d7/a0-a6 ; Riprendi i vecchi valori dei registri
109         RTS                      ; Torna all'ASMONE o al Dos/WorkBench
```

Ci sono solo quattro particolari aggiunti: uno è quello del controllo dopo l'apertura della `Graphics.library`, infatti se per qualche motivo non si potesse aprire, in `d0` anziché l'indirizzo del *GfxBase* troveremmo ZERO. Non si fa altro che uno pseudo `TST.L D0` e un salto alla label `EXIT` in caso di non apertura. Vedrete, con lo studio dei Condition Codes, come mai basti fare un `beq` dopo un `move`, senza usare il `tst`, per sapere se `d0` è azzerato. Un altro particolare è la comparsa della COPPER2 di sistema, (*GfxBase+$32*) che non è altro che il valore inserito in `$dff084`, `COP2LC`, dal sistema operativo; per ora non abbiamo mai usato la copperlist 2, ma in certe lezioni più avanti non mancheremo di illustrarne i casi di utilità. Altra "finezza" è quella di resettare gli sprites, ma solo se siamo sul kickstart 3.0 o superiori, dato che la funzione di reset degli sprite è disponibile da questa versione in avanti. La SubRoutine che resetta gli sprite è un classico esempio di programmazione "legale", con chiamate al sistema operativo... come potrete notare sbirciandola è più complicato usare il sistema operativo che programmare via hardware (non è vero??). Infine c'è il settaggio della priorità del task. Come sapete ogni programma che viene eseguito in multitasking ha una sua "priorità" rispetto agli altri. Ebbene, mettiamola al massimo! Ossia 127. In realtà non servirebbe, dato che dopo disabilitiamo del tutto il multitasking, ma vedremo in seguito che è utile settare la priorità al massimo e riabilitare il multitasking per caricare files dati da dischetto, harddisk o CDrom.

Con questa startup facciamo il possibile affinché il sistema operativo possa essere "scavalcato" senza problemi. Vediamo ora cosa possiamo fare per prendere in modo più completo il controllo dell'hardware di Amiga. Innanzitutto vanno introdotti i registri `DMACON`, `INTENA`, `ADKCON` ed `INTREQ`, che sono dedicati alla "chiusura" o "apertura" dei CANALI DMA, nonché alla abilitazione degli interrupt e di altre cose. Per ora nei listati abbiamo dato per scontato che il COPPER, I BITPLANES e gli SPRITE sono abilitati, infatti possiamo vedere sia i testi e i menù dell'ASMONE (BITPLANE) che la freccia puntatore del mouse (SPRITE). Questo significa che tali canali sono

abilitati. Comunque è meglio modificare di persona lo stato di questi registri per essere sicuri che i canali che ci interessano siano abilitati, e quelli che non ci interessano non lo siano. Come facciamo per le copperlist, basterà salvare lo stato di questi registri all'inizio, poi eseguire il nostro codice che abilita e disabilita a volontà, infine rimettere i registri nello stato di partenza, come nulla fosse accaduto. Ma innanzitutto vediamo cosa sono questi canali DMA. DMA significa "Direct Memory Access", ossia "accesso diretto alla memoria". Infatti nell'Amiga l'accesso alla memoria è molto complesso, dato che ci deve accedere non solo il processore, ma anche il copper per visualizzare immagini, il blitter per copiarle e spostarle, l'audio per suonare. Per evitare che succedano "incidenti" a tutti questi processori che vogliono mettere le mani sulla memoria (almeno quella CHIP) tutti contemporaneamente, è stato messo un sistema di "semafori" e di viadotti, si può parlare di urbanistica e viabilità. Infatti nel chip AGNUS esiste un gestore dei canali DMA, il quale coordina le operazioni, facendo accedere i chip custom e il 68000 alla memoria "a turno", quando il canale è libero. Tale accesso può essere sia di lettura che di scrittura (il copper LEGGE le copperlist, l'audio LEGGE le musiche, il blitter però SCRIVE anche le immagini, e così via). Esistono vari canali DMA, ognuno dedicato ad un particolare scopo, vediamoli:

DMA COPPER Attraverso questo canale il copper legge la COPPERLIST. Se viene disabilitato la copperlist non viene più letta, di conseguenza spariscono sia i bitplane, che gli sprite, che le eventuali sfumature fatte modificando il colore di sfondo varie volte mettendo dei `WAIT` in copperlist. In pratica lo schermo rimane di colore unito, del colore `COLOR0`. In questo caso si può cambiare il colore dello schermo solo col processore, con `MOVE.W #xxx,$dff180`.

DMA SPRITE Questo canale esegue il trasferimento delle strutture sprite, quelle puntate nei registri `SPRxPT` in copperlist. Abbiamo comunque già visto come sia possibile visualizzare sprites scrivendo direttamente nei registri `SPRxDAT`, facendo manualmente il lavoro del DMA. Disabilitando il solo canale del DMA SPRITE, spariscono gli sprite come se fossero puntati a zero, e rimangono sullo schermo i bitplane e le sfumature ottenute con i `WAIT` e `MOVE` della copperlist. Da notare che se viene disattivato il DMA BITPLANE, anche tenendo il DMA SPRITE attivo gli sprite spariscono.

DMA BITPLANE Disabilitando questo canale i bitplane puntati nei `BPLxPT` non vengono più visualizzati, in compenso, però, se è attivo il canale DMA del copper le eventuali sfumature fatte col `COLOR0` vengono visualizzate. Spegnere questo canale può equivalere a mettere ZERO bitplanes nel `BPLCON0`, ossia il `dc.w $100,$200` in copperlist. Da notare che se viene disabilitato il DMA BITPLANE, gli sprites spariscono assieme ai bitplane, anche se il canale DMA SPRITE è attivo. Questo succedeva anche quando mettevamo zero bitplanes nel `BPLCON0`.

DMA DISCO Serve per il trasferimento dei dati dal drive alla memoria CHIP in fase di lettura o scrittura.

DMA AUDIO1...DMA AUDIO4 Si tratta di 4 canali separati che controllano le 4 voci stereo dell'Amiga. Per esempio, per emettere un suono dalla voce 1, occore aprire il canale DMA AUDIO1, e per rendere muta tale voce basta richiudere tale canale DMA. Ovviamente i 4 canali sono sempre chiusi quando l'Amiga tace, ad esempio quando si utilizza il WorkBench senza musiche di sottofondo.

DMA BLITTER Questo DMA si occupa degli accessi in lettura e scrittura del blitter. Analizzeremo i canali DMA del blitter nella lezione dedicata a questo processore.

Ma come viene spartito il tempo di accesso alla memoria tra il processore e i chip custom? Dipende molto dalla risoluzione video e da quali canali sono abilitati. In pratica meno canali sono accesi e più vanno veloci il 68000 e gli altri CHIP in funzione. Vediamo il rapporto tra risoluzione video e DMA: l'immagine video è fatta di linee raster, cioè di linee disegnate dal pennello elettronico, che si chiama, appunto, raster. Sappiamo già come attendere una data linea verticale leggendo il `$dff006` (`VHPOSR`), oppure con la copperlist tramite un `WAIT`. Ebbene, in ogni linea raster sono possibili 227,5 accessi alla memoria, e il DMA ne utilizza solo 225. Un ciclo di accesso alla memoria, se vi interessa, ha la durata di 0,00000028131 secondi in uno schermo 320x256 PAL a 50Hz. Dato che il 68000 non farebbe in tempo ad accedere alla memoria ogni ciclo di BUS, gli sono concessi accessi solo durante i cicli pari, dunque 113 volte per linea raster. Il problema è che anche il Blitter ed il Copper possono accedere nei cicli pari, rubando cicli al povero 68000. I Cicli dispari sono utilizzati invece dal gestore DMA per gli accessi AUDIO, DISCO, SPRITE.

Riassumendo, ci sono 227/228 cicli per linea raster, divisi in cicli pari e cicli dispari. Nei 113 cicli dispari possono accedere alla memoria CHIP solo l'AUDIO, il DISCO, e gli SPRITE, a turno. Nei 113 cicli pari possono accedere alla memoria IL BLITTER, il COPPER e il 68000, a turno, dove però il povero 68000 ha poca priorità.

Capirete che se il blitter DMA è disattivato, il 68000 potrà accedere alla memoria più spesso, avendo più cicli pari liberi. Considerate che il DMA copper ha priorità sul DMA BLITTER, il quale a sua volta ha priorità sul 68000, che farebbe meglio a lavorare in FAST RAM. Infatti se il codice che il 68000 sta eseguendo è in FAST RAM anziché in CHIP RAM, il processore non subisce il minimo rallentamento. è per questo che conviene mettere il codice in fast ram con le `SECTION CODE`. Facciamo un esempio: se il copper stesse occupando il bus, sia il blitter che il 68000 dovrebbero attendere il prossimo ciclo pari. Il problema è che, mentre con una risoluzione di 320x256 LOWRES a 6 bitplanes il 68000 deve concedere "solo" la metà dei cicli pari al copper per visualizzare i 6 bitplanes, totalizzandone 56 per linea, nel caso di un 640x256 HIRES a 16 colori, ossia 4 bitplanes, il copper "ruba" quasi tutti i cicli pari al 68000, di conseguenza il programma rallenta (se non c'è FAST RAM sul computer). Gli accessi DMA durante la linea raster seguono un preciso schema: abbiamo visto che i cicli pari sono spartiti tra il COPPER, il BLITTER e il 68000. Nel caso dei cicli dispari, gli accessi per DISCO,AUDIO,SPRITE e BITPLANE seguono questo ordine: dalla linea orizzontale $7 alla $c avvengono gli accessi al DMA DISCO, dalla linea $D alla $14 quelli AUDIO, dalla $15 alla $34 quelli SPRITE, infine dalla $35 alla $e0 quelli per i BITPLANE. Riassumiamo:

```
             - MAPPA DEGLI ACCESSI DMA IN OGNI LINEA RASTER -

CICLI PARI: Sono 113, e sono spartiti tra Copper, Blitter e 68000, dove
            il copper ha la priorità maggiore, per cui se siamo in una
            risoluzione alta, es. 640x256 a 4 bitplanes, il 68000 non
            può accedere quasi mai alla memoria, causando un rallentamento
            molto evidente. L'unico rimedio è porre il codice in fast ram,
            per cui non ci sono rallentamenti al processore. Tra l'altro nei
            processori 68020 e superiori il codice in fast ram è sempre
            molto più veloce di quello in CHIP RAM.

CICLI DISPARI: Sono 113, e sono spartiti tra Audio, Disco e Sprite in questo
               ordine:
                        linea orizzontale:

                        \$07 - \$0C     Accesso al DMA DISCO
                        \$0d - \$14     Accesso ai 4 canali DMA AUDIO
                        \$15 - \$34     Accesso agli 8 canali DMA SPRITE
                        \$35 - \$e0     Accesso ai bitplane in memoria
```

In realtà ai fini della programmazione non serve sapere questi particolari tecnici, ma possono far capire come sia importante fare economia di canali DMA per raggiungere la massima velocità operativa. Se per esempio in una vostra produzione avete una schermata in HAM o in HIRES nella parte alta dello schermo, mentre sotto fate girare altre cose in bassa risoluzione, considerate che per il periodo dalla prima linea alla fine della schermata "impegnativa" come DMA (es. hires a 16 colori) sia il processore che il blitter subiscono rallentamenti, e potrebbero non farcela nel tempo rimasto sotto la figura. Per acquistare velocità potreste innanzitutto attivare i 16 colori solo dove effettivamente servono, esempio:

```
----------- inizio schermata, BPLCON0 settato per 16 colori HIRES
\ spazio NERO
/
 *** ######  #######  ####  ### ## # # # *##*       ### * ***#*##
 *** ######  #######  ####  ### ## # # # *##*       ### * ***#*## > FIGURA
 *** ######  #######  ####  ### ## # # # *##*       ### * ***#*##
\ spazio NERO
/
----- bplcon0 settato per risuluzione minore

 **
                                **      > PALLINE E CUBETTI 3d ROTANTI
               **
----- spazio nero


----- fine schermata, dc.w $ffff,$fffe
```

In questo caso vedete la cronaca di una copperlist. Supponiamo che alla routine 3d sotto la figura manchi proprio poco per essere eseguita in tempo per il cinquantesimo. Basta cambiare leggermente la copperlist e la routine potrebbe filare ad un fotogramma al secondo, vediamo cosa fare:

```
1  COPPERLIST
2          dc.w    $100,$200       ; 0 bitplanes nella zona "NERA" iniziale
3          dc.w    $3507,$fffe     ; aspetta la linea dove comincia la figura
4          dc.w    $100,$c200      ; attiva hires 16 colori
5          dc.w    $a007,$ffe      ; aspetta linea dove finisce la figura
6          dc.w    $100,$200       ; 0 bitplanes nella zona NERA sotto la pic
7          dc.w    $b007,$fffe     ; aspetta la fine della zona nera
8          dc.w    $100,$3200      ; 3 bitplanes lowres per routine vettoriale
9          dc.w    $e007,$fffe     ; sotto questa linea non arrivano le figure
10         dc.w    $100,$200       ; quindi spegnamo per bene il dma BITPLANE
11         ;       e magari facciamo una sfumatura col COLOR0 e i WAIT,
12         ;       per riempire la parte inferiore del monitor senza impegnare
13         ;       il DMA
14         dc.w    $ffff,$fffe
```

Per esagerare, potremmo anche restringere la finestra video dove le figure non riempiono in senso orizzontale tutto lo schermo. Facciamo questo caso: abbiamo un solido 3d che ruota al centro dello schermo, e abbiamo già chiuso il dma bitplane sopra e sotto di esso:

```
---------- inizio schermo, dc.w $100,$200



----------------   /\   ---- inizio solido, dc.w $100,$3200
                  / |\
                 /  | \
                /   |  \
                \___|__/
--------------------------- fine solido, dc.w $100,$200
```

```
------------ fine schermo, dc.w $ffff,$fffe
```

Come vedete, il solido ruota al centro dello schermo, e non occupa mai le zone di estrema destra ed estrema sinistra dello schermo. A questo punto, potremmo anche agire sul DIWStrt e DIWStop per “chiudere” un poco lo schermo, rendendolo largo solo il necessario, poi potremmo "riallargarlo" quanto serve per eventuali disegni più larghi sopra o sotto di esso:

```
1       dc.w    $8E,$2c81       ; DiwStrt LARGO normale per figura larga
2       dc.w    $90,$2cc1       ; DiwStop LARGO normale
3
4       WAIT
5
6       dc.w    $8E,$2c91       ; DiwStrt ristretto nella zona del solido
7       dc.w    $90,$2cb1       ; DiwStop
```

Infatti restringendo la finestra video risparmiamo tempo DMA, dato che il trasferimento dei bitplane avviene solo nella zona interna alla finestra video definita.

Chiudiamo questa parentesi, e vediamo come aprire e chiudere questi canali. Nell’ Amiga esiste un registro hardware ($dff096), denominato DMACon (=DMA Controller), che gestisce l’accensione di ogni singolo canale DMA. Il DMAConW ($dff096) serve solo per *scrivere* eventuali modifiche, mentre il DMAConR ($dff002) serve solamente per *leggere* i vari bit. Ecco la mappa dei 2 registri $dff096 e $dff002: (uguali ma uno per lettura e uno per scrittura). Il registro è *bitmapped* come il $dff100 (BPLCON0) per cui conta quali bit sono accesi o spenti, singolarmente:

```
(NOTA: i bits 13 e 14 sono a sola lettura (R), il 15 a sola scrittura (W))

DMACON ($dff096/$dff002)

bit- 15 DMA Set/Clear           (W)     (si può solo scrivere dal $dff096)
     14 BlitBusy (o BlitDone)   (R)     (si può solo leggere dal $dff002)
     13 Blit Zero               (R)     (si può solo leggere)
     12 X                       (non usato)
     11 X                       (non usato)
     10 BlitterNasty (BlitPri)  (R/W)   (R/W = Sia leggibili che scrivibili)
      9 Master  (DmaEnable)     (R/W) - "interruttore generale"
      8 DMA BitPlane (RASTER)   (R/W) - detto anche BPLEN
      7 DMA del Copper          (R/W) - detto anche COPEN
      6 DMA del Blitter         (R/W) - detto anche BLTEN
      5 DMA degli Sprite        (R/W) - detto anche SPREN
      4 DMA dei Dischi          (R/W) - detto anche DSKEN
      3 DMA Audio3 (voce 4)     (R/W) - ossia AUD3EN
      2 DMA Audio2 (voce 3)     (R/W) - ossia AUD2EN
      1 DMA Audio1 (voce 2)     (R/W) - ossia AUD1EN
      0 DMA Audio0 (voce 1)     (R/W) - ossia AUD0EN

*SET/CLR
-Il bit 15 è importantissimo: se esso è acceso allora i bit settati a 1 in
 scrittura nel \$96 servono ad accendere i relativi DMA, se il bit 15 è a 0,
 allora gli altri bit a 1 nel registro servono a spegnere i relativi canali.
 Mi spiego meglio: per accendere o spegnere uno o più canali è comunque
 necessario impostare ad 1 i relativi bit; quello che determina se quei
 canali devono venir spenti od accesi è il bit 15: se è ad 1 si accendono,
 mentre a 0 si spengono (sempre indipendentemente dal loro precedente stato).
 Diciamo che si sceglie su quali OPERARE, poi si decide se spegnere(0) od
 accendere(1) in base al bit 15.
```

Facciamo un esempio:

```
              ;5432109876543210
        move.w #%1000000111000000,$dff096       ; sono ACCESI i bits 6,7 e 8
```

```
               ;5432109876543210
        move.w #%0000000100100000,$dff096       ; sono SPENTI i bits 5 e 8.


N.B.: I BIT 14-10 RIGUARDANO IL BLITTER E I CICLI DI CLOCK DELLA CHIP,
      ARGOMENTO CHE VERRà AMPIAMENTE DISCUSSO IN SEGUITO.
      IN QUESTA LEZIONE NON SARANNO USATI.

*BlitBusy
-Il bit 14 è di sola lettura (lo si può leggere SOLO dal \$dff002), e serve a
 sapere se il blitter sta "blittando" (ossia lavorando) in quel momento.
 Questo bit è utilizzato per sapere se il blitter sta lavorando o meno,
 infatti, come diremo più avanti nel corso, non è possibile modificare i
 registri del blitter mentre sta ancora blittando... anzi, è possibile ma
 succederebbe il disastro ! Dunque bisogna attendere che questo bit sia a 0 con
 un btst prima di riutilizzare il blitter.

*Blit Zero
-Il bit 13 si imposta solo quando l' esito di una blittata è 0, ovvero,
 quando la RAM modificata con una qualsiasi blittata è stata interamente
 impostata a 0. Può verificarsi in molte situazioni, anche se torna comodo
 leggere questo bit solo in rare circostanze, per la verità(x es.: controllare
 se due oggetti -bob- collidono senza modificare la RAM), ma approfondiremo
 in seguito.

-I bit 12-11 non sono utilizzati dalla macchina, per il momento.

*BlitPri
-Il bit 10, se impostato, fa si che il blitter utilizzi tutti i cicli di bus
 della chip disponibili, "rubando" anche quei pochi che sono disponibili al
 povero 68000. Se questo accede alla Fast od alla ROM non verrà rallentato,
 altrimenti verrà addirittura fermato nell' accedere alla Chip.
 In pratica quando questo bit è a 1, il blitter ha una piena, anzichè
 completa, priorità sul 680x0

*DmaEn/Master
-Il bit 9 è l' interruttore generale: è necessario impostarlo ad 1 per
 far funzionare i DMA dei vari dispositivi. Lo si può spegnere ad esempio per
 disabilitare momentaneamente tutti i canali senza ricorrere all' azzeramento
 dell' intero registro.

-I bit 8-0 servono per accendere/spegnere i canali DMA dei vari dispositivi.
```

In sostanza solo i bit 10-0 sono switchabili (scambiabili) usando il bit 15. Per esempio, ora proviamo ad accendere solo i DMA dei plane, del copper e del blitter. Per fare questo, prima è necessario resettare il registro per spegnere tutti i canali, disabilitando quindi gli eventuali DMA indesiderati; poi si settano i DMA voluti:

```
        move.w  #$7fff,$dff096                  ; $7fff = %0111111111111111
                                                ; ossia: tutto spento: il
                                                ; bit 15 è a ZERO, dunque
                                                ; tutti gli 1 significano
                                                ; in questo caso SPENGI.
               ; 5432109876543210
        move.w  #%1000001111000000,$dff096      ; bits 6,7,8,9 settati, ossia
                                                ; BLITTER,COPPER,BITPLANE
                                                ; ed interruttore generale
                                                ; bit 15 ad 1, dunque tutti
                                                ; gli 1 significano ACCENDI
```

Il valore $7fff è %0111111111111111, quindi vengono resettati tutti i DMA-bit. Poi vengono impostati i DMA del Copper, dei Plane e del blitter, più quello master, grazie al bit 15 impostato ad 1 !

> *Il funzionamento di questo importantissimo registro è analogo a quello dei registri INTENA ed INTREQ, dunque non proseguite fino a che non avete più dubbi sulla funzione del bit 15 come "accendi/spegni" bit.*

Nei listati che abbiamo visto fino ad ora non sono stati mai usati i registri `$dff096` (`DMACON`) e `$dff002` (`DMACONR`), perché abbiamo dato per scontato che i canali DMA del copper, dei bitplanes e degli sprites fossero abilitati. In effetti, se al momento dell'esecuzione del programma si può vedere lo schermo dell'asmone, significa che sia il dma COPPER che quello BITPLANE è abilitato. La presenza della freccia puntatore indica che essa è visualizzata con il DMA SPRITE. Ma programmando a livello hardware non si possono fare compromessi, non si deve "sperare" che sia tutto come vogliamo. Abbiamo già visto come sia importante settare TUTTI i registri della copperlist come il `BPL1MOD`, il `DIWSTART/STOP` eccetera, per evitare di trovarseli con valori stani. Lo stesso faremo con i canali DMA: ne salveremo lo stato all'inizio della startup, poi li spegneremo tutti e accenderemo solo quelli desiderati, e alla fine rimetteremo i canali DMA nello stato iniziale, proprio come facciamo per la copperlist. Abbiamo detto che per leggere lo stato del `DMACON` occorre leggere dal `DMACONR`, cioè il `$dff002`. Una routine di "salvataggio" potrebbe essere:

```
1       move.w  $dff002,OLDDMA  ; DMACONR - salvo lo stato del DMA
```

Ora possiamo cambiarlo a nostro piacimento agendo sul `$dff096`, il registro per la scrittura:

```
1       move.w  #$7fff,$dff096  ; DMACON - azzero tutti i canali
2
3               ; 5432109876543210
4       move.w  #%1000001110100000,$dff096 ; Abilito Copper,Bitplane e Sprite
```

Niente di più facile. Ora dobbiamo rimettere a posto il vecchio valore, prima dell'uscita. Però ATTENZIONE! Non possiamo mettere `OLDDMA` direttamente nel `DMACON` (`$dff096`) così come lo abbiamo letto dal `DMACONR` (`$dff002`), perché il bit 15, quello `SET/CLR`, è di sola scrittura e in lettura è sempre a zero, dunque rimettendo il valore col bit 15 azzerato, i bit settati anziché accendere i canali DMA il spegnerebbero eventualmente. Serve dunque prima di settare il bit 15 del valore salvato in `OLDDMA`, in questo modo i bit settati varrebbero come ACCENSIONE. Ma come fare per settare il bit 15 di una word?? Ci sono infiniti modi. Uno sarebbe quello di usare l'istruzione `BSET`, ad esempio:

```
1       move.w  $dff002,d0      ; Salvo il DMACONR in d0
2       bset.l  #15,d0          ; setto il bit 15 (SET/CLR)
3       move.w  d0,OLDDMA       ; e salvo il valore in OLDDMA
4       ...
5       bsr.w   routines
6       ...
7       move.w  #$7fff,$dff096          ; azzero tutti i canali
8       move.w  OLDDMA(PC),$dff096      ; riattivo solo quelli che erano
9       rts                             ; attivi all'inizio.
```

Altrimenti si può usare l'istruzione `OR`. Ricordiamo il suo effetto sui bit:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

L'esempio di prima diventerebbe:

```
1       or.w    #$8000,OLDDMA   ; $8000 = %1000000000000000, ossia bit 15 ad 1
```

Come vedete dalla tabella sopra, i bit azzerati lasciano invariata la destinazione, in questo caso i primi 14 bit sono azzerati, dunque i primi 14 bit di OLDDMA dopo tale OR rimangono invariati (0 OR 0=0, 0 OR 1=1). Essendo il bit 15 settato, abbiamo che 1 OR 0=1, dunque viene settato il bit 15 e rimangono invariati gli altri 14 bit. Lo stesso che col BTST #15,d0. Nella startup conviene usare l'OR, perché sono salvati anche altri registri oltre al DMACON. Si tratta di INTENA ($dff09a in scrittura e $dff01c in lettura), INTREQ ($dff09c in scrittura e $dff01e in lettura) e ADKCON ($dff09e in scrittura e $dff010 in lettura). Per ora posso solo anticiparvi che tali registri sono bitmapped come il DMACON, e funzionano analogamente con il bit 15 che serve da SET/CLR. L'INTENA e L'INTREQ servono per gli interrupt, mentre ADKCON per compiti vari per il *disk drive* e l'*audio*. Vedremo come usare questi registri quando saranno trattati gli interrupt e l'audio, per ora salviamone lo stato assieme al DMACON. Vediamo ora come salvare questi 4 registri:

```
1       LEA     $DFF000,A5              ; Base dei registri CUSTOM per Offsets
2       MOVE.W  $2(A5),OLDDMA           ; DMACONR - Salva lo status del DMA
3       MOVE.W  $1C(A5),OLDINTENA       ; Salva il vecchio status di INTENA
4       MOVE.W  $10(A5),OLDADKCON       ; Salva il vecchio status di ADKCON
5       MOVE.W  $1E(A5),OLDINTREQ       ; Salva il vecchio status di INTREQ
```

Ora dobbiamo settare il bit 15 di tutte e 4 le word contrassegnate dalle label OLDDMA, OLDINTENA, OLDADKCON, OLDINTREQ, per poter ripristinare il valore all'uscita. Considerate che le 4 label sono messe consecutivamente:

```
1   OLDDMA:                 ; Vecchio status DMACON
2           dc.w    0
3   OLDINTENA:              ; Vecchio status INTENA
4           dc.w    0
5   OLDADKCON:              ; Vecchio status ADKCON
6           DC.W    0
7   OLDINTREQ:              ; Vecchio status INTREQ
8           DC.W    0
```

Dunque entra in gioco l'OR. se per una word basta fare un OR.w #$8000,dest possiamo sistemare con un solo OR 2 word, con un OR.L #$80008000,dest!!! In questo caso per 4 word bastano un paio di questi OR:

```
1       MOVE.L  #$80008000,d0           ; Prepara la maschera dei bit alti
2                                       ; da settare nelle word dove sono
3                                       ; stati salvati i registri
4       OR.L    d0,OLDDMA       ; Setta il bit 15 di tutti i valori salvati
5       OR.L    d0,OLDADKCON    ; dei registri hardware, indispensabile per
6                               ; rimettere tali valori nei registri.
```

Ecco che con poche istruzioni abbiamo salvato e "settato" tutti e 4 i registri che andremo ad azzerare subito dopo:

```
1       MOVE.L  #$7FFF7FFF,$9A(a5)      ; DISABILITA GLI INTERRUPTS & INTREQS
2       MOVE.L  #0,$144(A5)             ; SPR0DAT - ammazza il puntatore!
3       MOVE.W  #$7FFF,$96(a5)          ; DISABILITA I DMA
```

A questo punto possiamo abilitare i soli canali DMA che ci servono. All'uscita basterà azzerare tutti i registri e ripristinarli:

```
1       MOVE.W  #$7FFF,$96(A5)          ; DISABILITA TUTTI I DMA
2       MOVE.L  #$7FFF7FFF,$9A(A5)      ; DISABILITA GLI INTERRUPTS & INTREQS
3       MOVE.W  #$7fff,$9E(a5)          ; Disabilita i bit di ADKCON
4       MOVE.W  OLDADKCON(PC),$9E(A5)   ; ADKCON
5       MOVE.W  OLDDMA(PC),$96(A5)      ; Rimetti il vecchio status DMA
6       MOVE.W  OLDINTENA(PC),$9A(A5)   ; INTENA STATUS
7       MOVE.W  OLDINTREQ(PC),$9C(A5)   ; INTREQ
```

Niente di più semplice! Ora abbiamo il completo controllo dei canali DMA, e siamo sicuri che possiamo attivarli e disattivarli come ci pare, tanto all'uscita vengono ripristinati.

Per finire la nostra startup, potremmo definire un EQUATE. Ricordate cosa sono gli EQUATES? le direttive assembler `EQU` o =, che definiscono delle uguaglianze tra delle parole inventate a piacere e dei numeri, es:

```
CANE    EQU     10
GATTO   EQU     20

        MOVE.L  #CANE,d0        ; viene assemblato come MOVE.L #10,d0
        MOVE.L  #GATTO,d1       ; assemblato come MOVE.L #20,d1
        ADD.L   d0,d1           ; RISULTATO = 30
        rts
```

Le equates sono simili alle label, ma non terminano con i :. Al posto di `EQU` si può usare l'uguale (=):

```
CANE    =       10
```

Potremmo definire un `EQU` per i canali DMA da settare:

```
                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper e bitplane DMA abilitati
;                ----a-bcdefghij

;       a: Blitter Nasty    (Per ora non ci interessa, lasciamolo a zero)
;       b: Bitplane DMA     (Se non è settato, spariscono anche gli sprite)
;       c: Copper DMA       (Azzerandolo non è eseguita nemmeno la copperlist)
;       d: Blitter DMA      (Per ora non ci interessa, azzeriamolo)
;       e: Sprite DMA       (Azzerandolo spariscono solo gli 8 sprite)
;       f: Disk DMA         (Per ora non ci interessa, azzeriamolo)
;       g-j: Audio 3-0 DMA  (Azzeriamo lasciando muto l'Amiga)
```

Come vedete i bit 15 e il 9 devono essere **sempre settati**, dato che uno è il `SET/CLR` e l'altro il Master, l'interruttore generale. Nel listato si può mettere:

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane e copper
```

In questo modo possiamo avere all'inizio del listato l'`EQU` da modificare con sotto un breve sommario di aiuto con il significato dei bit.

Ma vediamo la startup, caricate in un buffer di testo `Lezione8a.s` e studiatelo. Nel commento finale sono riportate alcune note su certe modifiche minori.

Ora che abbiamo la startup "universale", possiamo anche metterla a parte in un file ed includerla all'inizio dei prossimi listati tramite la direttiva INCLUDE, che abbiamo già usato per includere la routine musicale. Basterà iniziare ogni listato con un:

```
        Section UsoLaStartUp,CODE

*****************************************************************************
        include "startup1.s"    ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************
```

Da notare che `Startup1.s` è la startup senza il `SECTION`, per cui dobbiamo mettere la direttiva `SECTION nome,CODE` o `CODE_C` ogni volta prima dell'include. La Startup fa un `BSR.S START`, per cui inizieremo il listato con:

```
START:
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA
```

Considerate che in `a5` è presente *$dff000*. In questo caso lo ho sfruttato.

Sembrerebbe una startup perfetta, ma gli manca ancora la ciliegina sopra. Questa ciliegina è la possibilità di lanciare il programma da icona del WorkBench senza problemi. Infatti, fino a che facciamo partire da cli/shell i nostri programmi basta questa startup, ma nel caso si vogliano disegnare delle icone per farli partire da WorkBench col doppio click del mouse occorre aggiungere qualche istruzione. è solo una formalità burocratica, ma se non si fa con programmi grandi, che allocano pure la memoria, può capitare che all'uscita non tutta la memoria sia liberata, o anche peggio. Ecco cosa occorre aggiungere all'inizio:

```
ICONSTARTUP:
        MOVEM.L D0/A0-A1/A4/A6,-(SP)    ; salva i registri nello stack
        SUBA.L  A1,A1
        MOVEA.L 4.w,A6
        JSR     -$126(A6)       ; _LVOFindTask(a6)
        MOVEA.L D0,A4
        TST.L   $AC(A4)         ; pr_CLI(a4) stiamo eseguendo dal CLI?
        BNE.S   FROMCLI         ; se si, salta le formalità
        LEA     $5C(A4),A0      ; pr_MsgPort
        MOVEA.L 4.W,A6          ; Execbase in a6
        JSR     -$180(A6)       ; _LVOWaitPort
        LEA     $5C(A4),A0      ; pr_MsgPort
        JSR     -$174(A6)       ; _LVOGetMsg
        LEA     RETURNMSG(PC),A0
        MOVE.L  D0,(A0)
FROMCLI:
        MOVEM.L (SP)+,D0/A0-A1/A4/A6    ; ripristina i registri dallo stack
        BSR.w   MAINCODE        ; esegui il nostro programma
        MOVEM.L D0/A6,-(SP)
        LEA     RETURNMSG(PC),A6
        TST.L   (A6)            ; Eravamo partiti dal CLI?
        BEQ.S   ExitToDos       ; se si, salta le formalità
        MOVEA.L 4.w,A6
        JSR     -$84(A6)        ; _LVOForbid - nota! Non serve il permit
        MOVEA.L RETURNMSG(PC),A1
        JSR     -$17A(A6)       ; _LVOReplyMsg
ExitToDos:
        MOVEM.L (SP)+,D0/A6             ; exit code
        MOVEQ   #0,d0
        RTS

RETURNMSG:
        dc.l    0
```

Non sto a commentare approfonditamente le chiamate alle librerie del sistema operativo, vi basti sapere che sono queste le formalità di cui parlavo. Se eseguite da workbench un programma che non ha questo codice all'inizio, il problema più grosso è quello che all'uscita da tale programma la memoria che ha occupato non viene liberata!!! Come potete notare, all'inizio si controlla se il programma è stato eseguito dal CLI o dal WorkBench, controllando un apposito flag di sistema. Se il programma è stato lanciato dal CLI, vengono saltate le formalità da seguire in caso di esecuzione da WB. Altrimenti,tali formalità sono eseguite. Anziché unire questo pezzo all'altra startup, conviene tenerla a parte, in modo da scegliere se includerlo o meno, dato che alcuni assemblatori, inclusa le versione di ASMONE modificato del corso, al momento dell'esecuzione causano un loop infinito, dato che "sembrerebbe" caricato da WorkBench, ma poi al momento dell'esecuzione delle "formalità" sembrerebbe il contrario. Altre versioni di Asmone o altri assemblatori invece eseguono tranquillamente questo codice, ma per compatibilità con ogni assemblatore si preferisce metterlo a parte:

```
;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

In questo modo, durante gli assemblaggi e le prove con *<J>* non lo includiamo, mentre prima di salvare l'eseguibile finale con *<WO>* lo facciamo includere.

Caricate la `Lezione8b.s`, il primo listato che utilizza la startup universale inclusa con INCLUDE. Comprende l'utilizzo sia dei bitplane che degli sprite, per cui potete fare delle prove per verificare l'abilitazione o meno dei canali DMA.

Vi siete spaventati trovando la NUOVA routine che attende la linea verticale? Ebbene, non c'è nulla di mostruoso, invece è molto migliore. Analizziamo la vecchia "routine":

```
1       cmp.b   #$xx,$dff006    ; VHPOSR
```

Ebbene, non facciamo che controllare il byte *$dff006*, che contiene la posizione verticale del pennello elettronico, i bit da 0 a 7, ossia da $00 a $ff. Ma come sapete dalla gestione dei WAIT in copperlist, il pennello elettronico supera la linea $FF, che in realtà non è che la 200 in uno schermo normale. Per raggiungere le posizioni oltre il $FF con i WAIT del COPPER abbiamo visto che dobbiamo aspettare la fine di tale zona:

```
1       dc.w    $FFDF,$FFFE     ; aspetta il limite della zona NTSC
```

Dopodiché il contatore riparte da $00

```
1       dc.w    $0007,$FFFE     ; aspetta linea $100
2       dc.w    $0107,$FFFE     ; aspetta la linea $FF+$01=$101
```

Fino a $38. Ebbene, anche il byte in *$dff006* si comporta in questo modo: una volta raggiunta la posizione $ff riparte da $00, indicando però $100, e continua fino a $138 (con $38), dopodiché riparte da $00, lo ZERO vero, per poi arrivare nuovamente a $ff, per fare gli altri $38, eccetera. è per questo che nei listati si attende sempre la linea $FF, o la $80, perché attendere la linea $00 o la linea $20 col *$dff006* avrebbe significato eseguire la routine 2 volte per fotogramma, in quanto $00 si verifica alla linea $00 e alla linea $100. Ma allora, come fare per aspettare tranquillamente le prime 38 linee, e le linee dopo la $ff? Insomma, serve una routine che aspetti senza errori una qualsiasi delle 312 linee della scansione. Non è difficile, dato che il bit alto, l'ottavo, si trova molto vicino al *$dff006*, esattamente al *$dff005*, il byte prima. Dobbiamo fare come abbiamo fatto con la posizione verticale degli sprite, infatti abbiamo il bit alto a parte. In questo caso, però, non si trova in giro per la memoria, ma proprio prima del byte in questione. Analizziamo la situazione:

```
$dff004 byte che ora non ci interessa, contiene il bit LOF per l'interlace
$dff005 ci interessa! il bit 0 è il V8, ossia il bit alto della pos. vertic.
$dff006 ormai lo conosciamo! i bits V7-V0, gli 8 bit bassi della pos. vertic.
$dff007 contiene la posizione orizzontale (H8-H1). La risoluzione è 1/160
        dello schermo. Ora non ci interessa proprio!!!
```

Il $dff004/$dff005 è il registro VPOSR, mentre il $dff006/$dff007 è il VHPOSR ogni registro infatti è lungo una WORD. Possiamo però accedere ad essi come singoli bytes, in certi casi. Per attendere la linea $100, possiamo fare così:

```
WaitVbl:
        btst.b  #0,$dff005
        beq.s   WaitVbl
```

Questa routine aspetta che il bit alto, il V8, sia settato. Se è settato significa che siamo alla linea $100, o comunque dopo di essa. Per fare una routine UNIVERSALE, possiamo fare così: (a5=$dff000)

```
1   Waity1:
2           MOVE.L  4(A5),D0        ; $dff004 e $dff006, ossia VPOSR e VHPOSR
3           LSR.L   #8,D0           ; sposta i bit di 8 posizioni a destra
4           AND.W   #%111111111,D0  ; Seleziona solo i bit della pos. verticale
5           CMP.W   #300,D0         ; linea 300? ($12c)
6           bne.s   Waity1
```

In questo caso abbiamo copiato *$dff004/5/6/7* in d0, poi spostiamo il tutto di 8 bit a destra, dato che i primi 8 bit di destra sono occupati dalla posizione orizzontale del *$dff007* che non ci interessa, portando la posizione verticale all'estrema destra. A questo punto, con un AND,

selezioniamo solo i primi 8 bit, ossia quelli del *$dff006* più quello alto del *$dff005*. In questo modo abbiamo in d0 il numero reale di linea da 0 a 312! Vi ricordo che il comando AND ha questo effetto:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

Infatti AND da come risultato 1 soltanto quando sia il bit del primo operando che quello del secondo operando sono ad 1, il comando si potrebbe tradurre con "SONO AD 1 SIA IL PRIMO CHE IL SECONDO BIT? SE SI, RISPONDO CON 1, SE NO INVECE RISPONDO CON UNO ZERO". Un AND infatti è utile per azzerare certi bit di un numero, nel nostro caso abbiamo azzerato quelli alti:

```
1       AND.W   #%00000000000000000000000111111111,d0
```

Forse appare più chiaro in esadecimale:

```
1       AND.W   #$000001FF,D0   ; solo il byte basso più il bit 8.
```

L'unico sconveniente è che servono ben 4 istruzioni. Vediamo di scrivere una routine che ne usi solo 3:

```
1 WBLANNY:
2       MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
3       AND.L   #$0001FF00,D0   ; Seleziona solo i bit della pos. verticale
4       CMP.L   #$00013000,D0   ; wait linea $130 (304)
5       BNE.S   WBLANNY
```

In questo caso lavoriamo su tutta la long senza spostare (shiftare) i bit. Basta ricordarsi che il numero di linea da attendere è spostato di 2 cifre a sinistra. Per esempio, per attendere la linea $FF:

```
1       CMP.L   #$0000ff00,D0   ; wait linea $130 (304)
```

è decisamente migliore, e più veloce. Vi consiglio di usare sempre questa routine. Altrimenti, se non vi dispiace "sporcare" un paio di registri in più, ci sarebbe una versione turbizzata, quella presente in Lezione8b.s:

```
1       MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
2       MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
3 Waity1:
4       MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
5       ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
6       CMPI.L  D2,D0           ; aspetta la linea $130 (304)
7       BNE.S   Waity1
```

Come vedete nella sostanza non cambia nulla, solo che le operazioni sono fatte tra registri anziché tra costanti, e questo è più veloce. La velocità è importante, perché se per esempio fate attendere la linea 50 con la prima routine che abbiamo visto che ha anche l'LSR, quando il processore ha finito tutti i test e si è accorto che siamo alla linea 50, siamo ormai alla metà della linea 50!!

Devo solo consigliarvi di fare attenzione che in a5, o a6, a seconda del registro che usate di base, ci sia sempre il $dff000, che non sia cioè sovrascritto da qualche subroutine. Per evitare ciò, potete salvare i registri col MOVEM come spiegato prima, oppure potete mettere un LEA $dff000,a5 sotto il MOVE.L #$1ff00,d1. Lo stesso vale per i registri d0, d1 e d2, dovete essere sicuri che non siano in uso da altri programmi, dato che li modificate.

Come ultima cosa, non attendete mai oltre la linea $138, dato che è l'ultima, o vi si bloccherà la routine in un loop infinito.

Se il video viene settato a frequenza NTSC (azzerando `$dff1dc`), il limite massimo è la linea $106.

Ora che abbiamo visto l'utilità delle istruzioni `AND/OR/LSR` per salvare i DMA e per controllare meglio la linea del VBLANK, veniamo a nuovi utilizzi di tali istruzioni logiche. Senza dubbio è il caso di analizzare come fare una routine di fade, ossia di dissolvenza di una figura dal nero, sfumando via via verso il colore pieno e lucente (e viceversa). Innanzitutto vediamo dove dobbiamo operare:

```
CopColors:
        dc.w $180,0,$182,0,$184,0,$186,0
        dc.w $188,0,$18a,0,$18c,0,$18e,0
        ....
```

In copperlist, dove sono i registri colore. Quello che dobbiamo fare è mettere al posto di quegli zeri i valori RGB (esattamente la word: $0RGB) giusti, "aumentandoli" in modo che con vari passaggi, uno per fotogramma, si innalzino fino a raggiungere i colori della nostra figura:

```
CopColors:
        dc.w $180,$000,$182,$fff,$184,$200,$186,$310
        dc.w $188,$410,$18a,$620,$18c,$841,$18e,$a73
        ...
```

Innanzitutto dobbiamo avere la lista dei colori della figura in una tabella da "consultare", altrimenti non sapremmo quando siamo "arrivati":

```
TabColoriPic:
        dc.w $fff,$200,$310,$410,$620,$841,$a73,...
        ...
```

(NOTA: il colore 0, il `$dff180`, in questa tabella non è stato messo, dato che è nero, $000, rimane sempre nero e non lo comprendiamo nella routine, partiamo invece dal colore 1, `$dff182`, che in questo caso è $FFF).

Per fare questa tabella basta togliere "a mano" i $180,$182,$184 dalla copperlist copiata con l'editor, e ovviamente rimangono i colori.

Ora che abbiamo la tabella con i colori "destinazione", come fare una routine che "aumenti" i colori fino a quelli giusti, nella tabella? Sicuramente dobbiamo lavorare separatamente per ognuna delle 3 componenti RGB, e per separarle possiamo usare l'`AND`, che come abbiamo visto "seleziona" solo una parte di bits, azzerando gli altri. Avendo l'indirizzo della tabella coi colori in `a0`, vediamo, per esempio, come separare la sola componente blu:

```
        MOVE.W  (A0),D4         ; Metti il colore dalla tabella colori in d4
        AND.W   #$00f,D4        ; Seleziona solo la componente blu ($RGB->$00B)
```

Ora in `d4` abbiamo solo il valore del BLU...se il colore era $0123, in `d4` dopo l'`AND.w #%000000001111,d4` (selezionati solo i 4 bits, o nibble), il valore è $0003, dunque siamo riusciti nell'impresa. Vediamo come selezionare la componente verde:

```
        AND.W   #$0f0,D4        ; Selez. solo la componente verde ($RGB->$0G0)
```

E quella rossa:

```
        AND.W   #$f00,D4        ; Selez. solo la componente rossa ($RGB->$R00)
```

Fin qui dovrebbe essere tutto chiaro. Ora, potremmo già fare la routine "FAKE", quella "FALSA", che funziona in questo modo: ogni volta si aggiunge #1 ad ogni singola componente, e si compara con il colore in tabella, se dobbiamo smettere di aggiungere a quella componente. Per esempio, se abbiamo il colore $0235 da raggiungere, avremo questi passaggi ogni fotogramma:

```
1)      $111    ; +$111, tutti e 3
2)      $222    ; +$111, tutti e 3
3)      $233    ; +$011, la componente RED va bene, aggiungo solo a Gr. e Blu
4)      $234    ; +$001, le componenti RED e GREEN vanno bene, +1 solo Blu
5)      $235    ; +$001, come sopra, +1 solo blu
```

Ogni volta dovremmo comparare con un CMP la componente ROSSA del colore in tabella con quello che stiamo "aumentando", se non la abbiamo raggiunta aggiungere 1, se la abbiamo raggiunta non aggiungere, poi fare lo stesso con il VERDE e il BLU, infine unire le 3 componenti "risultanti" tramite uno o più istruzioni OR, ottenendo la word colore risultante da mettere nella copperlist. E questo per ognuno dei 16, 32 colori o quanto sono. La quantità non è un problema per il processore, con dei cicli DBRA si può fare tutto. L'unico particolare è che il sistema che ho descritto non è esattissimo, e specialmente in AGA si vede che i colori vanno per i fatti suoi. Dunque, la struttura della routine rimane la stessa, ma dobbiamo cambiare il modo di calcolo. Dobbiamo prima notare una cosa: quanti fotogrammi, dunque quante volte dovremmo richiamare la routine per eseguire un fade completo? Se abbiamo il colore $0F3, un bel verde, per esempio, partendo da $000 e aggiungendo 1 ogni volta, con la routine di prima ci vorrebbero 15 `add.w #$010` per la componente verde, dato che deve raggiungere il valore $f (15). Dunque, consideriamo di fare una routine "parametrica", la quale possa calcolare i colori ad una delle 16 possibili FASI del fade, dove la fase 0 è il NERO completo, e la fase 16 è il colore pieno. Supponiamo di tenere "il conto" della fase da fare in una label `FaseDelFade`. Ogni volta dovremo fare:

```
1       addq.w  #1,FaseDelFade  ; sistema per la prossima volta la fase da fare
```

Dunque, al primo fotogramma faremo un `BSR.s Fade` con `FaseDelFade` ad 1, e i colori saranno messi scurissimi, il fotogramma successivo richiameremo la routine, ma con `FaseDelFade` a 2, e i colori si schiariranno (2/16 del colore pieno), infine quando la eseguiremo con `FaseDelFade` a 3, i colori saranno uguali a quelli della tabella. Prima riferendomi alla frazione, 2 sedicesimi del colore, stavo anticipando la tecnica da usare! Infatti mentre una routine Fake, uno degli orrori di routine fade che aggiungono semplicemente 1 ogni volta non sono precise frazionariamente, quella che faremo è accettabile. Veniamo al dunque: con la routine Fake avremmo questi passaggi per arrivare ad un $084:

```
$011
$022
$033
$044
$054
$064
$074
$084
```

Ebbene, quando si arriva a metà abbiamo un GRIGIO! $044!! anziché un verdino. In realtà, a metà, saremmo dovuti essere a $042, ossia al verdino scuro, che guarda caso è proprio 1/2 di $084. Ora, ecco che si affaccia la soluzione: disponendo del valore `FaseDelFade`, che possiamo chiamare MULTIPLIER, abbiamo che quando è a 0, dobbiamo calcolare 0/16 (zero sedicesimi) dei colori, ossia tutti ZERO. D'altronde, quando è ad 1, dobbiamo calcolare 1/16 dei colori. Così fino a 16/16, in cui il colore rimane uguale. Come implementare in istruzioni tale formula? Facile! Disponendo di una componente RGB isolata, ad esempio la BLU: (abbiamo in `d0` il MULTIPLIER)

```
1       MOVE.W  (A0),D4         ; Metti il colore dalla tabella colori in d4
2       AND.W   #$00f,D4        ; Seleziona solo la componente blu ($RGB->$00B)
3       MULU.W  D0,D4           ; Moltiplica per la fase del fade (0-16)
4       ASR.W   #4,D4           ; shift 4 BITS a destra, ossia divisione per 16
5       AND.W   #$00f,D4        ; Seleziona solo la componente BLU
6       MOVE.W  D4,D5           ; Salva la componente BLU in d5
```

In pratica moltiplichiamo per il MULTIPLIER la componente, poi la dividiamo per 16, in questo caso dividere per 16 equivale ad un `ASR.W #4,Dx`, come abbiamo già visto per la routine di print dei caratteri 8x8 che `MULU.W #8,Dx` si può sostituire da un `LSL.w #3,Dx`. Consideratelo

come una `DIVU.w #16,D4`, e tutto torna. Ripetendo 3 volte questo procedimento per le 3 componenti RGB, abbiamo la routine di FADE dal NERO ai colori, e se partiamo col multiplier a 16, sottraendo ogni volta #1 fino allo zero, avremo il fade contrario, dal colore al nero. Quest'ultimo si chiama FADE OUT, mentre il primo è il FADE IN.

Possiamo vedere in pratica il funzionamento della routine descritta nei due listati `Lezione8c.s` e `Lezione8d.s`. La differenza tra questi due listati è solo nell'ordine in cui vengono eseguite le operazioni di divisione delle 3 componenti RGB, ma il principio di moltiplicazione per il multiplier e divisione per 16 è lo stesso. La più chiara forse è quella di `Lezione8d.s`.

Il disegno è un logo del gruppo *RAM JAM*, fatto da FLENDER, che è italiano. Ho utilizzato questo disegno perché sono entrato a far parte di questo gruppo proprio mentre stavo scrivendo questa lezione. Dunque il corso da qua in avanti è una produzione *RAM JAM*!!!

Proseguiamo con una variazione sul tema, caricate `Lezione8e.s`. Questa è la stessa routine, con una lieve modifica che consiste nell'aggiunta di una componente dominante aggiuntiva, la quale fa assumere al disegno quella tonalità. Può essere utile per dare un'aria da carnevale al tutto.

Infine, sto per presentarvi una routine che può passare da qualsiasi colore a qualsiasi altro! In pratica occorrono due tabelle, una con i colori iniziali, per esempio se si partisse dal nero, una tabella con tanti zeri, e un'altra con i colori finali. Ne risulta che per fare il fade di prima, ossia dal nero ai colori normali, si deve mettere come prima tabella una tutta azzerata e come seconda quella coi colori, per passare dai colori al nero, (FADE OUT), come prima tabella si deve mettere quella coi colori normali, e come seconda una tutta azzerata.

A questo punto ecco le innovazioni: per esempio possiamo fare un fade dal BIANCO ai colori normali, mettendo come prima tabella tutti $FFF e come seconda quella coi colori normali. Esageriamo: possiamo passare da una colorazione ad un altra! Basta mettere nella prima tabella i colori come vogliamo che siano all'inizio, e come seconda come li vogliamo alla fine. In questo modo possiamo passare da una tonalità verde ad una bluastra, eccetera. Caricatevi `Lezione8f.s` e provate la routine, che mostra proprio gli esempi che ho fatto. Per quanto riguarda il funzionamento della routine, è piuttosto complicato, e non ho voglia di rivederla, se volete provare a capirla leggete i miei (pochi) vecchi commenti. Comunque almeno imparate ad usarla per i vostri scopi!

Ora voglio proporvi tre di listati, opera di altrettanti "alievi" partiti da zero con il mio corso, proprio come voi, incoraggiante no?

LEZIONE8g.s parallasse 10 livelli (di Federico Stango)

LEZIONE8h.s pannello di controllo con gadgets (di Michele Giannelli)

LEZIONE8h2.s scrolltext 8X8 (di Lorenzo Di Gaetano)

Questi tre listati utilizzano le sole conoscenze del disco 1 del corso. Io ho cambiato solamente la startup, inserendo la `startup1.s` al posto del vecchio modo di inizializzare del disco1. Spero che anche voi stiate facendo delle prove "autonome", altrimenti a che serve leggersi tutto come un romanzo? Sveglia!!! E se avete fatto qualcosa di carino vedete di spedirmelo, almeno lo metto nelle prossime lezioni e diventate famosi come Fiorello.

Proseguiamo ora con una domanda frequente: "Come funzionano gli equalizzatori presenti nella piccola demo `AMIGAET.EXE` del disco 1 del corso?". Ebbene, ho "tagliato" quel pezzo di listato, potete vedere come funziona il tutto in `lezione8i.s`.

Attenzione: la routine `music.s` del disco 2 non è la stessa di quella del disco 1. Le 2 modifiche sono la rimozione di un BUG che alle volte causava una guru all'uscita del programma, e il fatto

che mt_data è un puntatore alla musica, e non LA musica. Questo permette di cambiare la musica più facilmente, per creare music disks, come si vede in lezione8i2.s.

Siamo arrivati a fare gli equalizzatori, ma non abbiamo ancora visto come stampare un punto, ossia "plottare un dot". Rimediamo subito con Lezione8l.s. (Poi mettere plottata di diversi planes da 3d_stars.s)

Ok, ora che sappiamo stampare i punti, stampiamone tanti uno accanto all'altro per fare delle "linee", in Lezione8m.s e Lezione8m2.s.

Ebbene, se si possono fare linee, si possono fare anche curve paraboliche, basta moltiplicare X*X, in Lezione8m3.s, Lezione8m4.s, Lezione8m5.s.

Vediamo ora come "ottimizzare" la routine di stampa dei punti. Come avete visto, ha una moltiplicazione, il che è molto male perché le moltiplicazioni sono lente. Come fare per "toglierla"? Dobbiamo moltiplicare per 40, dunque basta "eseguire" tutte le moltiplicazioni possibili, cioè i primi 256 multipli di 40, e scrivere i risultati in una tabella. Ora abbiamo in questa tabella tutti i "risultati" della moltiplicazione in questione a seconda dei vari casi. Basta fare in modo che venga "preso" dalla tabella il risultato giusto ogni volta, come prendiamo la X o la Y giusta dalle tabelle delle coordinate per gli sprite. Vediamolo in pratica in Lezione8n.s.

Verifichiamo se effettivamente la nuova routine è più veloce di quella vecchia, scrivendo e cancellando tutto lo schermo, in Lezione8n2.s

Dato che abbiamo visto come azzerare un punto (basta mettere un BCLR al posto del BSET), proviamo ad "animare" un punto come abbiamo fatto per gli sprite, scrivendolo e cancellandolo ogni frame a posizioni diverse, in Lezione8n3.s.

Provate a farvi versioni modificate, a più biplanes, con più di un punto alla volta, eccetera. Per stampare su 2 bitplanes, ossia 4 colori, potete fare così: color0 è lo sfondo, mentre abbiamo 3 colori diversi per plottare. Considerando di avere i 2 bitplanes con i nomi Bitplane1 e Bitplane2, potreste farvi 3 routines, una che plotta nel bitplane1, una che plotta nel bitplane2, e una che plotta in entrambi i bitplanes, e saltare ad una di queste 3 routines per stampare in uno dei 3 colori.

Incredibile! Lorenzo di Gaetano ha scritto al volo un suo listato! vedetevelo: Lezione8n4.s

Mi immagino che abbiate fatto un programma che studia funzioni megacomplesse, che disegna onde come la sigla di Quark. Allora si può fare un breve stacco pubblicitario per i wait del copper, che non sono stati usati per le routines dei punti. Date un'occhiata a cosa possono fare dei semplici wait e color0, senza l'ausilio di nessun bitplane, in Lezione8o.s. Non ci sono trucchi, solo che la copperlist viene "costruita", oltre che modificata, ecco la routine che "crea" il pezzo saliente della copperlist:

```
; INITCOPPER crea la parte di copperlist con tanti WAIT e COLOR0 di seguito

INITCOPPER:
        lea     barcopper,a0    ; Indirizzo dove creare la copperlist
        move.l  #$3001fffe,d1   ; Prima wait: linea $30 - WAIT in d1
        move.l  #$01800000,d2   ; COLOR0 in d2
        move.w  #coplines-1,d0  ; numero di linee copper
initloop:
        move.l  d1,(a0)+        ; metti il WAIT
        move.l  d2,(a0)+        ; metti il COLOR0
        add.l   #$02000000,d1   ; prossimo wait, aspetta 2 linee più in basso
        dbra    d0,initloop
        rts
```

Come si può vedere, il risultato di questa routine è di creare:

```
barcopper:
        dc.l    $3001fffe       ; wait linea $30
        dc.l    $01800000       ; color 0
        dc.l    $3201fffe       ; wait linea $32
        dc.l    $01800000       ; color 0
        dc.l    $3401fffe       ; wait linea $34
```

```
7       dc.l    $01800000       ; color 0
8       ....
```

Pensate quanto spazio e quanto tempo risparmiamo in questo modo.

Per terminare la lezione, credo sia il caso di trattare una caratteristica del processore che, nonostante sia importantissima, fino ad ora non è stata discussa. Infatti **credevate** di sapere abbastanza sul 68000, ma in realtà fino ad ora è stato studiato "all'acqua di rose", il minimo indispensabile per fare delle routines. Infatti non sono stati nominati i Codici di Condizione, e con essi il CCR, il quale fa parte dello SR (Status Register). Ecco i 16 bit che compongono il registro:

```
SR:

                            ---
15      T - TRACE             \
14      - non usato dal 68000  |
13      S - SUPERVISOR         |
12      - non usato dal 68000  |- SYSTEM BYTE
11      -                      |
10      I2 \                   |
9       I1  > INTERRUPT MASK   |
8       I0 /                ___/
7       -                     \
6       -                      |
5       -                      |
4       X - EXTENSION          |- USER BYTE (Condition Code Register)
3       N - NEGATIVE           |  (contiene i flag aritmetici)
2       Z - ZERO               |
1       V - OVERFLOW (eccesso) |
0       C - CARRY (RIPORTO)  ___/
```

Ebbene, questo misterioso registro contiene dei bit riguardanti i FLAG di condizione, per la precisione il suo byte basso, detto CCR (Condition Code Register) contiene tali FLAG. Il byte alto dello SR lo tratteremo più avanti, quando parleremo di INTERRUPT e di MODO SUPERVISORE. Per ora posso solo anticiparvi che il processore può funzionare in due modi, uno *UTENTE* (USER) e uno *SUPERVISORE*. Normalmente i programmi che scriviamo sono eseguiti in modo UTENTE. Quando ci serviranno gli interrupt vedremo come passare da modo Supervisor a modo User e viceversa, ma ricordatevi che alcune istruzioni si possono eseguire solamente in modo SUPERVISORE, se provate ad eseguirle in modo USER va tutto in coma profondo. Queste istruzioni sono dette PRIVILEGIATE, fate attenzione! Per ora ci basterà CAPIRE il byte basso dello SR, il CCR.

Ogni istruzione, agendo, può influenzare i flag, per esempio se una sottrazione da un risultato negativo si setta il flag N, se da zero si setta il flag Z, se una addizione ha come risultato un numero più grande, ad esempio, di quello contenibile in D0.1, si setterà il bit V, overflow, che indica l'impossibilità di contenere il risultato nella destinazione. Questo vale anche per il Carry, ossia il riporto, che si setta in caso di riporto. Si potrebbero controllare i flag stessi testando il byte CCR, ma siccome il 68000 è il miglior processore del mondo, sono presenti istruzioni a sufficienza per sapere lo stato dei flag: si tratta di Bcc, dove cc sta per Condition Codes e può essere sostituito con CS,EQ,GE,GT,HI,LE,LS,LT,MI,PL... Vi ricordate che parlando del modo di funzionamento delle istruzioni CMP seguite da BEQ e BNE giustificammo il fatto che il BEQ/BNE sapesse come era andata al CMP perché il risultato del CMP veniva scritta su un "foglietto"? Ebbene, il "foglietto" dove il CMP scrive il risultato per il BEQ/BNE è il CCR, il byte basso di SR!! In realtà questo foglietto è composto da 4 bit, più un quinto, detto eXtend, che serve a scopi particolari. Tramite questi 4 bit, si possono creare un bel pò di "situazioni", non solo BEQ e BNE, ma si può sapere se un numero è più grande o più piccolo di un altro, se due numeri sono uguali, se si verifica un riporto in una operazione, se il risultato è negativo, eccetera. Ecco tutti i Bcc:

```
1                bhi.s   label   ; > per numeri senza segno
2                bgt.w   label   ; > per num. con segno
3                bcc.s   label   ; > detto anche BHS, Carry = 0 (senza segno)
4                bge.s   label   ; >= per num. con segno
5                beq.s   label   ; = per tutti i numeri
6                bne.w   label   ; >< per tutti i numeri
7                bls.w   label   ; <= per num. senza segno
8                ble.w   label   ; <= per num. con segno
9                bcs.w   label   ; < per num. senza segno; detto anche BLO,
10                               ; significa che il Carry = 1
11               blt.w   label   ; < per num. con segno
12               bpl.w   label   ; Se Negative = 0 (PLus)
13               bmi.s   label   ; Se Negative = 1, (Minus) num. con segno
14               bvc.w   label   ; V=0, no OVERFLOW (risultato contenibile)
15               bvs.s   label   ; V=1 OVERFLOW (risultato troppo grande per
16                               ; essere contenuto nella destinazione)
```

Ora vediamo come usare i `Bcc` dopo `CMP.x OP1,OP2`

```
1        beq.s   label   ; OP2 =  OP1 - per tutti i numeri
2        bne.w   label   ; OP2 >< OP1 - per tutti i numeri
3        bhi.s   label   ; OP2 >  OP1 - senza segno
4        bgt.w   label   ; OP2 >  OP1 - con SEGNO
5        bcc.s   label   ; OP2 >= OP1 - senza segno, detto anche *"BHS"*
6        bge.s   label   ; OP2 >= OP1 - con SEGNO
7        bls.w   label   ; OP2 <= OP1 - senza segno
8        ble.w   label   ; OP2 <= OP1 - con SEGNO
9        bcs.w   label   ; OP2 <  OP1 - senza segno, detto anche *"BLO"*
10       blt.w   label   ; OP2 <  OP1 - con SEGNO
```

Ed ora come usarli dopo un `TST.x OP1`

```
1        beq.s   label   ; OP1 =  0 - per tutti i numeri
2        bne.w   label   ; OP1 >< 0 - per tutti i numeri
3        bgt.w   label   ; OP1 >  0 - con SEGNO
4        bpl.s   label   ; OP1 >= 0 - con SEGNO (oppure BGE)
5        ble.w   label   ; OP1 <= 0 - con SEGNO
6        bmi.w   label   ; OP1 <  0 - con SEGNO (oppure BLT)
```

Come si vede dopo un `CMP` si possono sapere un bel pò di cose! Si possono notare i segni > (maggiore), >= (maggiore o uguale), =, >< (diverso), <= (minore o uguale), < (minore), e per di più esiste un Bcc di queste comparazioni per numeri normali, e uno per i numeri Signed (con segno). Per quanto riguarda i numeri negativi, fino ad ora abbiamo solo accennato che, ad esempio, -1 è $FFFFFFFF, -5 è $FFFFFFFB, stabilendo più o meno che il bit alto, ossia il 31 se siamo in longword, il 15 se in .w e il 7 se in .b, è quello del segno, ossia che se è ad 1 il numero è negativo, e procede come se tornasse "indietro" da $FFFF che è -1, a $FFFE che è -2, $FFFD per -3 eccetera, fino ad arrivare (in campo .w) a $8001, cioè -32767, seguito da $8000, cioè -32768, che è il numero più negativo possibile in una word con segno, e corrisponde a %1000000000000000, ossia il bit alto del segno settato e gli atri tutti azzerati: eravamo partiti da -1, cioè %111111111111111. Questo sistema usato per avere numeri negativi in binario è detto complemento a due. Sappiamo già che il bit più significativo, ossia quello più a sinistra, rappresenta il segno: se = 0 è positivo, se = 1 è negativo. Questo sistema vale sia per numeri .byte (il bit è il 7), che per quelli .word (il bit è il 15), che per quelli .longword (il bit è il 31). Vediamo ora in dettaglio come funziona il complemento a due: abbiamo notato che non basta cambiare il bit più significativo per passare da positivo a negativo, facciamo l'esempio di +26 e -26 in campo .word:

```
          ;5432109876543210
+26       %0000000000011010       ($001A)
-26       %1111111111100110       ($FFE6)
```

Il bit 15 in +26 è azzerato e in -26 è settato, ma non è evidentemente l'unica modifica da fare per passare da -26 a -26!!! Occorre fare il complemento a due di %0000000000011010, che

consiste nell'*invertire* tutti i bit e *aggiungere* 1 al risultato. Proviamo se è vero: invertendo tutti i bit si ottiene:

```
%1111111111100101
```

Aggiungiamo 1:

```
%1111111111100101 +
                1 =
-----------------
%1111111111100110
```

Se vi confonde la fila di 1, isolate i 6 bit bassi: %100101 è 25, se si aggiunge 1 = %100110, cioè 26, con i bit dal 7 al 15 tutti ad 1, cioè -26. Se vogliamo -26 in un byte, basta %11100110, ossia $E6. Se vogliamo -26 in una long: %11111111111111111111111111100110 = $FFFFFFE6 Si può scegliere di usare i nostri byte, word o long come vogliamo, con segno o senza segno, dipende dalle istruzioni che usiamo e dal nostro programma. Per chiarire, ecco quanto può contenere un `.b`, un `.w` o un `.l` a seconda del sistema usato, se "normale" o "complemento a 2":

```
BYTE con segno   .8 bit  - da -128 ($80) a +127 ($7f)
BYTE senza segno .8 bit  - da 0 ($00) a 255 ($ff)
WORD con segno   .16 bit - da -32768 ($8000) a +32767 ($7fff)
WORD senza segno .16 bit - da 0 ($0000) a 65535 ($ffff)
LONG con segno   .32 bit - da -2147483648 ($80000000) a +2147483648 ($7fffffff)
LONG senza segno .32 bit - da 0 ($00000000) a 4294967299 ($ffffffff)
```

Come si può vedere, nel campo SIGNED BYTE i numeri da 128 a 255 sono considerati come i valori da -128 e -1, in campo SIGNED WORD i valori che vanno da 32768 a 65535 sono considerati come i valori da -32768 e -1. Lo stesso valore per la notazione `.longword`. Ricapitolando, ecco 2 sistemi per ottenere un numero negativo dal positivo:

Sistema 1: Dato il numero N=%00110 (6 decimale) per trovare -N si esegue la negazione bit a bit di N, ottenendo N=%11001 (-7 decimale) e poi si somma 1 al risultato:

```
N=%11001+%00001=%11010 (-6 decimale)
```

Sistema 2: Dato il numero N=%00110 (6 decimale) per trovare -N si esegue la negazione bit a bit di N fino all'1 meno significativo N=%11010 (-6 decimale).

Se nella nostra routine non accade mai di scendere sotto lo zero, è bene usare un byte per i suoi 255 valori, se invece volessimo andare da -50 a +50, occorre usare istruzioni come `BGT`, `BLE`, `BLT`, che confrontano numeri con segno, al posto di `BHI` e `BLS`, ad esempio, che confrontano numeri senza segno. Addizioni e sottrazioni funzionano sia con numeri con segno che senza segno, mentre le moltiplicazioni e le divisioni no, infatti sono presenti due tipi di istruzione per numeri con o senza segno: `MULU` e `DIVU` per i numeri senza segno, `MULS` e `DIVS` per i numeri con segno.

Chiariti i numeri negativi, vediamo i bit del `CCR`, ossia i flag, uno ad uno:

bit 0 - Carry (C) settato ad 1 quando il risultato di un'addizione genera un riporto ('carry'), o quando un sottraendo è maggiore del minuendo, cioè quando una sottrazione ha richiesto un "prestito". Il bit di Carry contiene inoltre il bit più o meno significativo di un operando sottoposto ad uno shift o ad una rotazione. Viene posto a zero quando l'ultima operazione eseguita non ci sono né riporti, né "prestiti". Per esempio, un modo per settare il flag CARRY è questo:

```
1       move.l  #$FFFFFFFF,d0
2       ADDQ.L  #1,d0
```

Il risultato è d0=00000000, con il flag CARRY e ZERO settati, perché abbiamo ecceduto il massimo contenibile in .l, e il risultato è pure ZERO!

bit 1 - Overflow (V) viene settato se il risultato dell'ultima operazione tra numeri dotati di segno il risultato è troppo grande per poter essere contenuto nell'operando destinazione, ad esempio se tale risultato supera i limiti -128...+127 nel campo byte. Ad esempio, la somma.b 80+80 genera un oVerflow, avendo superato +127. In campo .w i limiti sono -32768...+32767, e in campo .l sono -/+ 2 miliardi. Da notare che la somma 80+80 in campo byte non setta il flag di Carry ed eXtend, ma solamente quello di oVerflow, dato che 160 non supera 255, il massimo contenibile in un byte per numeri normali.

bit 2 - Zero (Z) settato quando l'operazione genera il risultato zero (utile anche per controllare il decremento di un contatore), nonché quando si confrontano due operandi uguali.

bit 3 - Negative (N) viene settato ad 1 se in una operazione il bit alto del numero, in formato complemento a due, è settato. In pratica se il risultato è un numero negativo questo bit è settato, altrimenti azzerato. Il complemento a due si ottiene facendo il complemento a uno dell'operando (ossia invertendo tutti i bit), aggiungendo quindi 1; ad esempio, +26 in binario è %000110010; il suo complemento a uno è %11100101 (inversione dei bit 0 in bit 1 e viceversa); aggiungendo 1 si ottiene %11100110. Il bit 7, detto bit di segno, viene copiato nel bit 3 dello Status Register; Nel caso di -26, ad esempio, N viene settato, indicando un numero negativo.

bit 4 - Extend (X) è una ripetizione del bit di Carry, e viene usato in operazioni effettuate in notazione BCD (Binary Coded Decimal: il numero decimale 20, ad esempio, non viene rappresentato con 00010100, ma nella forma due decine, zero unità 0010 0000) ed in operazioni binarie "estese" come ADDX e SUBX, versioni particolari delle istruzioni di addizione e sottrazione ADD e SUB.

Alla luce di queste nuove conoscenze, vedetevi il testo di riferimento su tutte le istruzioni del processore, con relativi effetti sui FLAG del CCR, il 68000-2.TXT, un'"evoluzione" del vecchio 68000.TXT del primo disco, che ormai è roba da bambini per voi (o no?).

Prima di iniziare la LEZIONE9.TXT, sarebbe bene che vi leggeste tutto il 68000-2.TXT, almeno sarete veramente ferrati sulle istruzioni della CPU! Consideratelo come una LEZIONE8b,TXT, "*fatevelo*" tutto, carpitene l'essenza. Ammetto che può *spaventarvi* (se siete mezze cartucce) leggerlo tutto, ma una volta presa confidenza con quello che è scritto in quel bel testo da 100K potrete finalmente dire in giro che sapete programmare il 68000. Tra l'altro, se più avanti trovate istruzioni che non conoscete, non potete lamentarvi, perché sono spiegate nel 68000-2.TXT!!

Come prima cosa, andatevi a vedere le istruzioni CMP e Bcc, in cui i vari tipi di Bcc sono spiegati più diffusamente, poi partite dall'inizio e arrivate alla fine, magari rileggendolo più volte, facendo delle pause tra una lettura e l'altra, mangiandovi un panino. Questo 68000-2.TXT è il secondo macigno che dovete superare; il primo era la LEZIONE2.TXT dove avete imparato le prime basi, gli indirizzamenti. Molti si sono fermati a quella collina. Ora vi si presenta una montagna, e altrettanti non avranno il fegato per superarla. Ma chi la supererà, potrà cercare di arrivare alla vetta!

Lo avete letto almeno una volta? Avete chiari i *Condition Codes*? Ecco degli esempi che verificheranno se avete capito. Sono stati gentilmente scritti da Luca forlizzi (the Dark Coder) e da Antonello Pardi (Deathbringer), permettendomi di velocizzare la scrittura delle lezioni sull'AGA e sul 3d.

```
Lezione8p1a.s   -> CC nell'istruzione MOVE
Lezione8p1b.s   -> CC in MULU/MULS
Lezione8p1c.s   -> CC in DIVU/DIVS
Lezione8p2a.s   -> CC e registri indirizzi Ax
Lezione8p2b.s   -> Estensione del segno nei registri indirizzi Ax
Lezione8p3.s    -> CC in TST
Lezione8p4.s    -> CC in AND,NOT,OR,EOR
Lezione8p5.s    -> CC in NEG
Lezione8p6.s    -> CC in ADD
Lezione8p7.s    -> CC in CMP
Lezione8p8.s    -> CC in ADDX
Lezione8p9.s    -> CC in lsr,asr,lsl,asl
```

Per finire caricate il mio `Lezione8p9b.s`, che contiene anche un "quesito".

Prima di passare alla prossima lezione, ci sarebbero un paio di cose che vorrei dirvi. Quel mio amico che programma l'avventura, Michele, mi ha chiesto delle cose l'ultima volta che mi è venuto a trovare, e suppongo che potrebbero interessare anche a molti di voi. Lui ha fatto un pannello di controllo in basso, simile a `Lezione8h.s`, e nella parte alta visualizza le varie figure, che carica dal dischetto (vedremo più avanti come caricare files con la libreria di sistema `dos.library`). Il problema è che aveva i `.raw` delle figure, ma la palette di ogni figura la doveva tenere nel programma principale in tabelle preparate, una per figura, e una routine si occupava di copiare i colori della tabella giusta in copperlist a seconda della figura caricata. Questo però ingarbugliava il codice, dato che le figure sono dozzine. Allora mi sono ricordato che con gli iffconverter, compreso il KefCon, si può salvare ANCHE LA PALETTE in fondo al .RAW! Basta cambiare l'opzione CMAP OFF in BEHIND, e in fondo al .raw viene attaccata la palette, dal color0 all'ultimo, word dopo word. Si potrebbe anche scegliere BEFORE, che attacca la palette prima della pic, ma in tal caso occorrerebbe puntare a "dopo la palette", saltandola. Stabilito che conviene salvare con la CMAP BEHIND (in fondo), vediamo cosa cambia nel file .raw salvato.

Il file è uguale, ma più lungo, nel caso del logo di questa lezione è più lungo di 16 words, infatti ha 16 colori `.w` in fondo, come in questo esempio (per capirci):

```
1  inizio_pic:
2          incbin   'logo320*84*16c.raw'     ; bitplanes.raw normali
3          dc.w $000,$fff,$200,$310          ; palette
4          dc.w $410,$620,$841,$a73
5          dc.w $b95,$db6,$dc7,$111
6          dc.w $222,$334,$99b,$446
7  fine_pic:
```

Ho opportunamente risalvato il logo in questo formato, vediamo con quale semplice routine si può copiare la palette in copperlist, in `Lezione8q.s`. Da notare che la pic se puntata normalmente funziona anche nei listati precedenti, infatti abbiamo solo delle word "in più" che non sono visualizzate essendo "dopo" la fine dell'ultimo bitplane.

Altra cosa che mi è stata chiesta, è: come si fa a sapere quale processore e quale kickstart è presente sulla macchina? In `Lezione8r.s` questo mistero è svelato... basta consultare degli appositi bit dedicati allo scopo!

Dunque, se siete convinti di aver capito ogni cosa fino a qui, potete passare a caricare la `LEZIONE9.TXT`, che vi introdurrà FINALMENTE al blitter, che a questo punto sicuramente stavate domandandovi se veramente esiste.

Una nota: se sapete leggere l'inglese, vi sarà certamente utile avere questi libri fondamentali:

- La seconda edizione del manuale dell'hardware di Amiga: "Amiga Hardware Reference Manual" codice ISBN: 0-201-18157-6
- PER QUANTO RIGUARDA IL 680x0:

- Motorola, “MC68020 32-bit Microprocessor User Manual, fourth edition”, Prentice Hall ISBN 0-13541657-4
- Motorola, “MC68030 Enhanced 32-bit Microprocessor User Manual, second edition” Prentice Hall ISBN 0-13-566951-0, Motorola ISBN 0-13-566969-3.
- Motorola, “MC68040 32-bit Microprocessor User Manual”

Forse non vi conviene prendere l’user manual del 68000, né quello del 68040, dato che il 68000 è spiegato (spero) abbastanza bene nel `68000-2.txt`, e il 68040 per ora è appannaggio di pochi fortunati, dunque demo o giochi che vanno su solo 68040 avrebbero poca diffusione. Inoltre le differenze maggiori ci sono tra 68000 e 68020, mentre tra 68020 e 68030 le differenze sono poche, lo stesso vale per il 68030 rispetto al 68040. Le maggiori differenze comunque sono nella MMU e nelle istruzioni di controllo delle CACHE, ma programmando demo e NON sistemi operativi ciò non ci interessa più di tanto.

CAPITOLO 9

LEZIONE 9 - IL BLITTER

Autori: Luca Forlizzi, Alvise Spanò, Fabio Ciucci

(Directory `Sorgenti5`) - quindi scrivere *`<V Assembler2:sorgenti5>`*

9.1 IL BLITTER

In questa lezione inizieremo a parlare del blitter. Chiunque possieda un Amiga avrà sicuramente sentito parlare di questo speciale circuito posto all'interno del suo computer che risulta esserne uno dei maggiori punti di forza qualora lo si confronti con altri computer. Non tutti però sanno che cosa il blitter sia in realtà e per quali motivi sia così tanto utile. In effetti la maggioranza degli effetti speciali che potete ammirare nelle demo (come per esempio gli scrolltext sinusoidali o i vectorballs) fanno uso del blitter. E allora, vi chiederete, come mai si possono realizzare questi effetti anche sui dei PC che non hanno il blitter? Il motivo è che in realtà tutto ciò che il blitter può fare, potrebbe essere fatto con il microprocessore, ed è appunto in questo modo che fanno i PC. Il blitter, però è in grado di svolgere i suoi compiti in maniera molto più veloce, in certi casi anche 10 volte più veloce. È grazie al blitter che effetti speciali che con un PC si possono realizzare solo se si ha a disposizione un 386 veloce o addirittura un 486, mentre sono ordinaria amministrazione per un Amiga 500 il cui processore (68000 a 7Mhz come sapete bene) è molto più lento dei 386 e 486. Quindi capirete che per chi voglia programmare demo o giochi sull'Amiga, la conoscenza del blitter sia indispensabile. Cominceremo lo studio delle capacità del blitter partendo dalle più semplici, che a prima vista potrebbero sembrare misere, ma che scopriremo via via nascondere la potenza che ha permesso la creazione dei giochi e delle demo più spettacolari. Occorre però notare che i programmi scritti per 68020+ spesso tendono ad usare la CPU anziché il blitter, dato che quest'ultimo non aumenta di velocità.

9.2 Funzioni del blitter

La parola *blitter* è un'abbreviazione di *BLock Image TransferER* ovvero "copiatore di blocchi di immagine". Il blitter è dunque uno strumento che ci permette di spostare "pezzi" di immagini. In realtà, come scoprirete in seguito, questa è solo una delle capacità del blitter, che è in grado di

effettuare anche operazioni più complesse. Come sapete, un immagine all'interno dell'Amiga è costituita semplicemente da una zona di memoria che contiene i dati che definiscono il colore di ogni singolo pixel. Se non vi ricordate bene come sono formate le immagini è bene che andiate a ripassare le lezioni 4 e 5 prima di proseguire oltre. Quando il blitter effettua un operazione su un "pezzo" di immagine, lavora in realtà sulla zona di memoria che forma il "pezzo" di immagine in questione. In effetti il blitter opera semplicemente su zone di memoria, indipendentemente dal fatto che esse contengano un'immagine grafica, un suono o il codice di un programma. Questo significa che il blitter può essere utilizzato anche in compiti che non riguardano la grafica. È importante precisare, però che il blitter, al pari del copper dei circuiti audio e di tutto il resto dei chip "custom" dell'Amiga, non è in grado di operare su tutta la memoria disponibile, ma solo su una parte di essa denominata "chip ram".

Per accedere alla memoria il blitter utilizza i canali DMA di cui si è parlato in termini generici nella lezione 8, a cui vi rimando in caso di dubbi. Il blitter ha a disposizione ben 4 canali DMA, di cui 3 (denominati A, B e C) servono per *leggere* dati dalla RAM (e per questo vengono detti canali "sorgente") mentre il quarto (canale D) serve per *scrivere* nella memoria (e pertanto è detto canale "destinazione"). Come tutti i canali DMA, quelli del blitter trasferiscono una word di dati alla volta.

Lo schema generale di un'operazione blitter (detta "BLITTATA") è molto semplice: il blitter, attraverso i canali A,B e C, legge dati dalla memoria, effettua delle operazioni su di essi e scrive i risultati in memoria attraverso il canale D. Per eseguire una blittata è dunque necessario specificare le seguenti informazioni:

1. quali canali usare per questa operazione
2. che operazione effettuare sui dati letti
3. per ogni canale usato, l'indirizzo da dove iniziare a leggere e scrivere
4. quanti dati leggere o scrivere

Notate che la quantità di dati letta (o scritta) durante un'operazione è la stessa per tutti e quattro i canali: se in un'operazione uso i canali A, B e D, il numero di words che vengono lette attraverso il canale A è uguale al numero di words che vengono lette attraverso il canale B e al numero di words che che vengono scritte attraverso il canale D.

Queste informazioni vengono specificate attraverso alcuni registri hardware. I registri che controllano il blitter sono, come tutti i registri hardware, a 16 bit. Vi sono però molti registri che hanno indirizzi consecutivi. Questo fatto rende possibile accedervi a coppie utilizzando delle `move.l` invece che delle `move.w`, analogamente a quanto abbiamo visto per le coppie di registri `BPLxPT` (*$dff0e0...*) e `COPxLC` (*$dff080...*).

Prima di iniziare a scrivere nei registri, è necessario però essere certi che il blitter sia fermo, cioè che non stia già compiendo un'altra operazione. è indispensabile aspettare che l'ultima "blittata" sia finita prima di farne un'altra, altrimenti si potrebbero causare esplosioni e crolli nel raggio di 100 metri, un vero cataclisma, paragonabile ad un bombardamento aereo.

Per sapere se il blitter è fermo o sta "blittando", basta controllare lo stato di un bit (il bit 6) del registro `DMACONR` (*$dff002*). Se tale bit vale 1 allora il blitter sta lavorando, mentre se vale 0 vuol dire che il blitter ha finito. In pratica, quindi basta una semplice istruzione assembler:

```
1 AspettaBlit:
2 btst    #6,$dff002      ; dmaconr – il blitter ha finito?
3 bne.s   AspettaBlit     ; Non andare avanti fino a che non ha finito
```

Purtroppo, a complicare le cose c'è un fastidiosissimo BUG hardware nelle prime versioni del chip Agnus (il chip che contiene il blitter) a causa del quale la prima volta che viene effettuata

una lettura del bit in questione si ha un risultato sbagliato: bisogna effettuare una lettura a vuoto prima di poter conoscere con esattezza lo stato del bit.

Dopo esserci assicurati che il blitter sia fermo, possiamo scrivere nei registri le informazioni che gli sono necessarie per la blittata e che abbiamo elencato sopra.

Vediamo ora in dettaglio come si procede.

1. Per ogni blittata possiamo abilitare o disabilitare indipendentemente i canali DMA, in modo da usare solo quelli che ci interessano, mediante dei bit di abilitazione che, se vengono settati a 1, abilitano il canale; se invece vengono azzerati lo disabilitano. I bit di abilitazione si trovano nel registro di controllo BLTCON0 (*$dff040*):

canale	nome bit di abilitazione	posizione del bit in BLTCON0
A	SRCA	8
B	SRCB	9
C	SRCC	10
D	DEST	11

2. Per specificare quale operazione effettuare si usano i bit da 0 a 7 del registro di controllo BLTCON0, detti *MINTERMS*. Il valore che assumono tali bit determina l'operazione effettuata dal blitter. Il funzionamento dei MINTERMS è abbastanza complicato, e lo spiegheremo in dettaglio in seguito.

3. Vediamo ora come indicare gli indirizzi di partenza dei canali. Ad ogni canale è connesso un puntatore alla chip RAM che serve appunto per memorizzare l'indirizzo di partenza di un'operazione. Durante l'operazione il valore contenuto nel puntatore varierà automaticamente, indicando di volta in volta l'indirizzo della word che il blitter legge o scrive. Un puntatore è costituito (come per i canali DMA degli sprites e dei planes) da una coppia di registri a 16 bit, uno che contiene i 16 bit meno significativi (cioè più bassi) e uno che contiene i rimanenti (alti). In questa tabella sono riassunti i nomi e gli indirizzi dei puntatori:

canale	registro alto		registro basso	
nome	indirizzo		nome	indirizzo
A	BLTAPTH	$DFF050	BLTAPTL	$DFF052
B	BLTBPTH	$DFF04C	BLTBPTL	$DFF04E
C	BLTCPTH	$DFF048	BLTCPTL	$DFF04A
D	BLTDPTH	$DFF054	BLTDPTL	$DFF056

   Chiaramente, queste coppie di registri possono essere trattate come singoli registri a 32 bit – come per i puntatori alle CopperList ed ai Plane –, e, quindi, essere scritti con una singola istruzione `move.l` all'indirizzo di BLTxPTH. Pertanto d'ora in avanti li considereremo come singoli registri a 32 bit, usando i nomi di BLTxPT e riferendoci agli indirizzi *$dff050*, *$dff04c*, *$dff048* e *$dff054* (salvo eventuali eccezioni che verranno opportunamente segnalate).

   I registri di puntatore dovrebbero essere scritti con un indirizzo in byte, ma poiché il blitter lavora solo su WORDS, il bit meno significativo del nostro indirizzo viene ignorato, dunque occorre ricordare che gli indirizzi devono essere PARI, ossia allineati a WORDS. Quindi occorre ricordarsi che si possono scrivere solo indirizzi PARI della memoria CHIP, sia per le sorgenti che per la destinazione.

   NOTA: Assegnate i bit non usati a zero, specialmente quelli che non hanno nessuna funzione nemmeno in ECS, dato che in versioni future potrebbero essere usati per chissà quali scopi e i risultati sarebbero imprevedibili.

4. L'ultima operazione da effettuare è indicare la quantità di dati che devono essere letti o scritti. Ciò viene fatto tramite il registro BLTSIZE (*$dff058*). Questo registro permette al blitter di considerare i dati che legge e scrive non come una semplice sequenza di word, ma come una sorta di rettangolo bidimensionale composto da words. Per esempio il blitter considera una sequenza di 8 words, come un rettangolo largo 8 words e alto 1 linea:

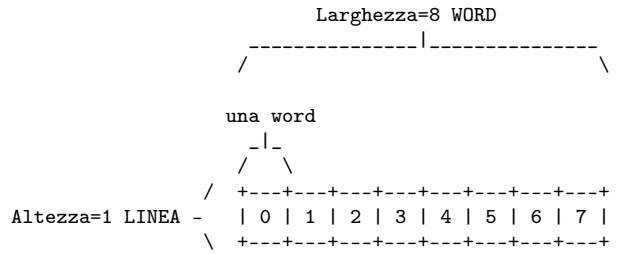


fig. 1 rettangolo di words 8*1

Facciamo un altro esempio: una sequenza di 50 words può essere considerata come un rettangolo di 10 word X 5 linee:

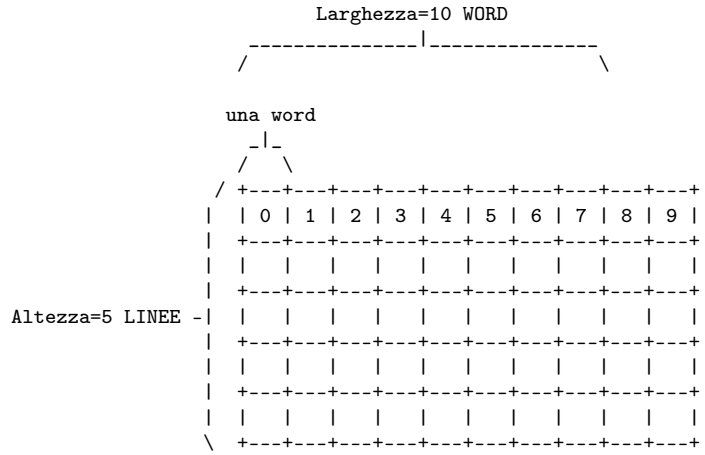


fig. 2 rettangolo di words 10*5

Questo fatto che a prima vista può sembrare un inutile complicazione è in realtà una delle caratteristiche che rendono il blitter tanto potente. Tra un attimo vedremo bene per quale motivo. Prima però vediamo come funziona BLTSIZE. Per specificare la quantità di dati coinvolta nella blittata, si scrivono in BLTSIZE le dimensioni del rettangolo di words che i dati formano. Nei 6 bit bassi va espressa la dimensione orizzontale, ovvero il *numero di word* che costituisce ogni linea orizzontale; nei 10 bit alti va espresso *numero di linee* orizzontali che costituiscono il rettangolo: in sostanza, nei 6 bit bassi va la larghezza in X del rettangolo, nei 10 bit alti va l' altezza in Y del suddetto rettangolo. È da notare che se il valore dei 10 bit alti (altezza) è 0, il blitter blitterà 1024 linee, e se il valore dei 6 bit bassi (larghezza in word) è 0, il blitter blitterà 64 word: la più grande blittata, dunque, si ottiene scrivendo `move.w #$0000,$dff058`. Essa sarà di 64 word X 1024 linee (=64*2*1024=128 Kb). Il registro BLTSIZE ha inoltre un'altra importantissima funzione: **scrivendoci si attiva il blitter**, dando inizio all'operazione specificata. Per questo motivo, **si deve scrivere nel registro** BLTSIZE **sempre dopo aver scritto in tutti gli altri registri del Blitter**, altrimenti

la blittata inizierà prima che voi abbiate settato correttamente tutti i registri, producendo risultati differenti da quelli voluti.

A questo punto è bene mettere in pratica quello che si è appreso sinora, guardando alcuni esempi. In questi esempi vengono utilizzati anche dei registri di cui non abbiamo ancora parlato, come `BLTDMOD` e `BLTCON1`. Per il momento ignorateli, li spiegheremo in seguito.

In `Lezione9a1.s` vedrete come cancellare una zona di memoria usando il blitter. Per eseguire un'operazione di cancellazione è necessario usare solo il canale D, in quanto l'unica cosa che dobbiamo fare è scrivere delle word azzerate nella memoria. Disabilitando il canale sorgente, nella destinazione sarà scritto il valore $00. Inoltre per definire un'operazione di cancellazione è necessario scrivere il valore $00 nei MINTERMS, cioè nei bit 0-7 (il byte basso) del registro BLTCON0.

In `lezione9a2.s` invece useremo il blitter per copiare dei dati da una zona di memoria in un'altra. Per questa operazione useremo i canali A e D. I dati verranno letti dalla memoria tramite il canale A e verranno scritti tramite il canale D. Per definire un'operazione di copia dal canale A al canale D è necessario scrivere il valore $F0 nei MINTERMS.

9.3 Prima applicazioni del blitter

Ora inizieremo ad usare il blitter in applicazioni grafiche. Sappiamo che un'immagine è costituita da word di dati in memoria. Poiché mediante il blitter possiamo compiere operazioni sulla memoria, modificando i dati che costituiscono un'immagine provochiamo una modifica dell'immagine stessa. Facciamo quindi un breve ripasso sulla rappresentazione delle immagini, limitandoci per ora al caso di un singolo bit-plane.

Un bit-plane è un insieme di word, ognuna delle quali rappresenta lo stato di un pixel: una word rappresenta 16 pixel disposti orizzontalmente. La prima word del bit-plane rappresenta i 16 pixel più a sinistra della prima riga dell'immagine. Le word seguenti rappresentano in ordine tutti i pixel della prima riga. Quando sono finiti i pixel della prima riga, si comincia allo stesso modo con quelli della seconda. Se su una riga ci sono ad esempio 320 pixel, sono necessarie 320/16=20 word per rappresentarla tutta; quindi le prime 20 word del bit-plane rappresentano la prima riga dell'immagine,le word dalla 21-esima alla 39-esima rappresentano la seconda riga, eccetera:

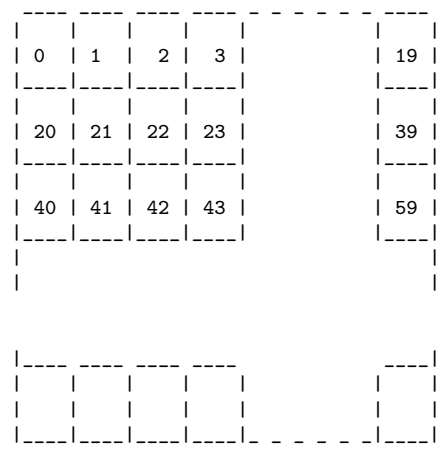


Fig. 3 Rappresentazione in memoria di un'immagine:
ogni quadrato è una word

Abbiamo visto che con il blitter possiamo copiare dati da un punto all'altro della memoria. Se copiamo dati all'interno di un bit-plane, i valori da noi copiati verranno usati per formare l'immagine sullo schermo. Il blitter, poiché come abbiamo detto lavora su dati di dimensione WORD (16 bit), ci permette di modificare l'immagine a gruppi di WORD, cioè a gruppi di 16 pixel. Per esempio, se con il blitter scriviamo sopra la 21-esima word del bitplane mostrato in figura, modificheremo i 16 pixel più a sinistra della seconda riga dell'immagine. Supponiamo ora di avere un'immagine alta una sola riga e larga un certo numero L di pixel. Proprio per il fatto che il bit-plane è diviso in word, che contengono 16 pixel, è conveniente che la larghezza in pixel della nostra immagine, cioè L, sia un numero multiplo di 16, in modo che l'immagine sia contenuta esattamente in L/16 word. Ciò può essere ottenuto aggiungendo dei pixel di valore 0 alla fine della nostra immagine, come illustrato dal seguente esempio:

Questa è un'immagine larga 20 pixel e alta una sola riga.

```
11001101010100011001
\__________________/
         |
     20 pixel
```

Non è comoda da gestire perché 20 non è un multiplo di 16. Allora aggiungiamo dei pixel di valore 0 alla fine in modo da rendere la larghezza pari a 32 pixel, cioè pari ad un multiplo di 16.

```
11001101010100011001000000000000
\______________________________/
         |
    32 pixel
```

La nostra immagine è memorizzata tra i dati nel nostro programma. Per farla apparire sullo schermo, dobbiamo copiarla nella zona di memoria dedicata al bit-plane. L'immagine assumerà sullo schermo una posizione corrispondente alle word del bit-plane nelle quali la copieremo. Supponiamo di voler disegnare l'immagine sullo schermo in maniera che il primo pixel di essa, cioè il pixel più a sinistra, assuma le coordinate X e Y (vi ricordo che il sistema di coordinate dello schermo ha l'origine, cioè il punto di coordinate X=0 e Y=0, nell'angolo superiore sinistro, le coordinate X crescono andando verso destra mentre le Y crescono andando verso il basso). Tale pixel sarà contenuto in una word del bit-plane.

Per il momento limitiamoci a considerare il caso in cui X sia anch'esso multiplo di 16. Ciò ci assicura che il nostro pixel sia il primo (cioè il pixel più a sinistra) della word a cui appartiene. In questo modo, una volta calcolato l'indirizzo di tale word, potremo copiarci (con il blitter) la prima word dell'immagine. Le altre word che formano la nostra immagine verranno copiate naturalmente nelle successive word del bit-plane.

Tutto questo, visto che il blitter è in grado di copiare sequenze di word, si può fare con una singola blittata che abbia come indirizzo sorgente la prima word dell'immagine, e come indirizzo destinazione, l'indirizzo della word del bit-plane a cui appartiene il pixel di coordinate X e Y. Vediamo come si fa per calcolare questo indirizzo.

Numeriamo le word del bit-plane a partire da 0, come mostrato in figura, e calcoliamo il numero della word che ci interessa: da tale numero risaliremo poi all'indirizzo vero e proprio.

Cominciamo col calcolare il numero della prima word della riga Y, ricordando ancora una volta che ogni riga è formata da 20 word e che le righe sono numerate a partire da 0. Potete notare dalla figura che la prima word della riga 0 (la prima riga) ha numero 0, la prima word della riga 1 (la seconda riga) ha numero 20, la prima word della riga 2 ha numero 40, la prima word della riga 3 ha numero 60 e così via.

In generale quindi, la prima word della riga Y ha numero Y*20. I numeri delle altre word della riga sono consecutivi a quello della prima: la seconda word della riga ha numero Y*20+1, la terza word della riga ha numero Y*20+2 e così via.

Possiamo chiamare "distanza" di una certa word R dalla prima word della riga cui R appartiene, la quantità che bisogna aggiungere al numero della prima word della riga per ottenere il numero della word R: in pratica, poiché la seconda word della riga ha numero Y*20+1, diciamo che essa ha "distanza" 1 dalla prima word della riga; allo stesso modo la terza word della riga, che ha numero Y*20+2 ha distanza 2 dalla prima word della riga, e così via. Possiamo dire inoltre che la prima word della riga ha distanza 0 da se stessa. È molto semplice calcolare la distanza tra la word che contiene il pixel di coordinata X e la prima word della riga, come vedremo aiutandoci con la seguente figura:

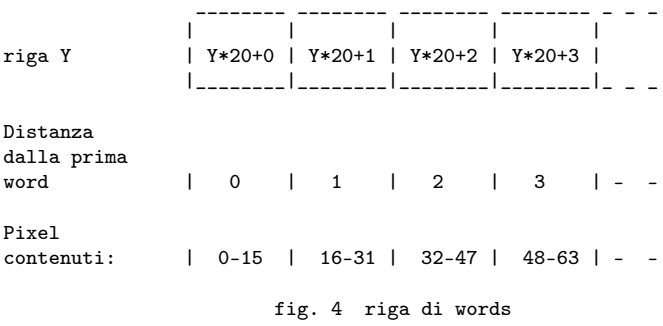


fig. 4 riga di words

La coordinata X del nostro pixel rappresenta la distanza (in pixel) tra esso e il primo pixel della riga. Poiché ogni word contiene 16 pixel, la prima word di una riga contiene i primi 16 pixel della riga, cioè quelli che hanno una coordinata X (=una distanza dal bordo) da 0 a 15. La seconda word invece contiene i pixel la cui coordinata X varia da 16 a 31, la terza word i pixel con X che varia da 32 a 47 e così via: ogni 16 pixel abbiamo una word.

Quindi per calcolare la distanza tra le word, basta dividere la distanza in pixel (cioè il valore di X) per 16. Poiché abbiamo scelto X multiplo di 16, il risultato sarà un intero. Per esempio, se X=32, la distanza in word vale 32/16=2. Infatti, come vedete nella figura il pixel 32 della riga Y è il primo pixel della seconda word della riga, il cui numero è proprio Y*20+1. Con lo stesso calcolo vediamo che il pixel che ha X=64 è contenuto nella word che dista 64/16=4, word il cui numero è Y*20+3. Questo calcolo funziona anche se X=0: infatti abbiamo distanza 0/16=0 cioè la word di numero Y*20+0 che è appunto la prima word della riga.

In totale, quindi la word che contiene il pixel X,Y è la word con numero N dato dalla seguente formula:

```
N=(Y*20)+(X/16)
```

Questa formula è valida per bit-plane nei quali una riga è formata da 20 word. In generale la formula è:

```
N=(Y*NUMERO_WORD_CHE_FORMANO_UNA_RIGA)+(X/16)
```

Dal numero della word possiamo risalire all'indirizzo corrispondente: basta conoscere l'indirizzo della prima word del bit-plane e aggiungerci il numero della word moltiplicato per 2 (la moltiplicazione è necessaria perché l'indirizzo è espresso in byte e 1 word = 2 byte):

```
Indirizzo word=(Indirizzo bitplane)+N*2 .
```

Nell'esempio `Lezione9b1.s` troverete l'applicazione di tutto quanto abbiamo detto. Nell'esempio `Lezione9b2.s` vedrete una serie di blittate in diverse posizioni dello schermo.

Iniziamo ora ad occuparci di immagini che abbiano altezza maggiore di una riga. Abbiamo visto quando abbiamo parlato del registro `BLTSIZE`, come il blitter consideri i dati su cui deve operare come dei "rettangoli" di words. Questa caratteristica è molto utile, perché gli consente di lavorare agevolmente con immagini rettangolari. Supponiamo ad esempio di voler copiare all'interno di un bitplane un'immagine larga 32 pixel e alta 2 linee. Questa piccola immagine andrà ad occupare una piccola porzione del bitplane, evidenziata nella figura dalle linee oblique.

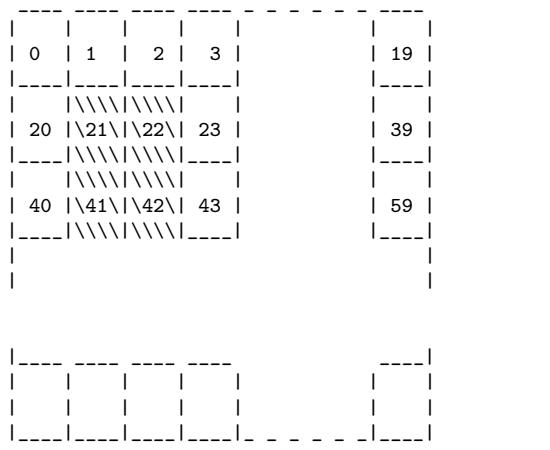


Fig. 5 Un bit-plane con evidenziata la porzione sulla quale blitteremo

Si tratta di un piccolo rettangolo largo 2 words (cioè 32 pixel) e alto 2 linee. Capirete immediatamente che per effettuare la copia è necessario specificare in `BLTSIZE` le dimensioni del rettangolo. Ma ciò non è sufficiente. Per rendercene conto, mettiamoci per un momento nei panni del blitter e proviamo a eseguire noi la copia, puntando la nostra attenzione per il momento solo sulla fase di scrittura.

Sappiamo (perché è scritto in `BLTDPT`) l'indirizzo della word in alto a sinistra del rettangolo (la word 21 nella figura). Inoltre sappiamo (è scritto in `BLTSIZE`) le dimensioni del rettangolo da copiare. Molto bene. Leggiamo la prima word e la copiamo all'indirizzo della word 21. Ora dobbiamo copiare la seconda word della prima riga. Sappiamo che questa word è consecutiva alla prima word, quindi aggiungiamo 2 all'indirizzo della prima word (che è scritto in `BLTDPT`) e sappiamo l'indirizzo della seconda word da scrivere. La scriviamo e abbiamo finito la prima riga. Molto soddisfatti ci prepariamo a scrivere la seconda riga. E qui ci accorgiamo che c'è un piccolo problema: la prima word della seconda riga **non è consecutiva** all'ultima word della prima riga! Infatti come potete vedere dalla figura l'ultima word della prima riga è la word 22 mentre la prima della seconda riga è la word 41.

Come facciamo a calcolare l'indirizzo della prima word della seconda riga? Nella figura è rappresentato un bitplane largo 20 word, ma è solo un esempio. Come fa il povero blitter a sapere quante word è largo il bitplane? Infatti potremmo essere in presenza di un bitplane più largo dello schermo visibile! Anzi a pensarci bene chi l'ha detto al blitter che lo stiamo usando per copiare un rettangolo sullo schermo? E se invece stessimo semplicemente copiando dei dati in una copperlist? È evidente che da solo il blitter non sa trarsi d'impaccio. Ma non c'è problema, lo aiutiamo noi.

Quello che al blitter serve di sapere è semplicemente come fare per calcolare l'indirizzo della prima word di una riga sapendo l'indirizzo dell'ultima word della riga precedente. Se guardate un attimo la figura vi convincerete il blitter deve semplicemente "saltare" le word da 23 a 40 comprese. Ciò può essere fatto aggiungendo all'indirizzo della word 22 (cioè l'indirizzo dell'ultima word della prima riga, che il blitter già conosce) il numero di bytes di differenza rispetto alla word 42 (che è appunto la prima word della nuova riga). Tale numero di bytes, che si chiama MODULO, è ovviamente uguale al numero di words da "saltare" moltiplicato per 2 (poiché come ben sapete una word occupa 2 bytes).

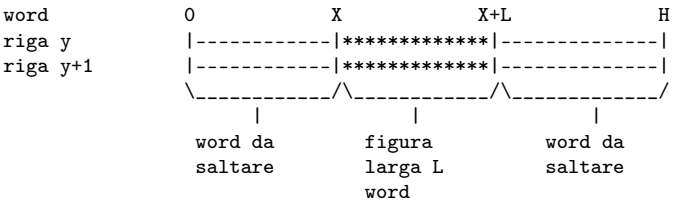


Fig. 6 Modulo

In generale, se dobbiamo copiare un rettangolo largo *L* words all'interno di una bitmap larga *H* words, il MODULO espresso in bytes si ottiene con la seguente formula:

```
MODULO = (H-L)*2
```

Il calcolo H-L ci darebbe il modulo espresso in words, la moltiplicazione per 2 serve per esprimerlo in bytes. Nel nostro esempio il MODULO vale (20-2)*2. Se vi ricordate avevamo già incontrato il concetto di modulo relativamente ai bit-planes. Il modulo del blitter funziona esattamente allo stesso modo.

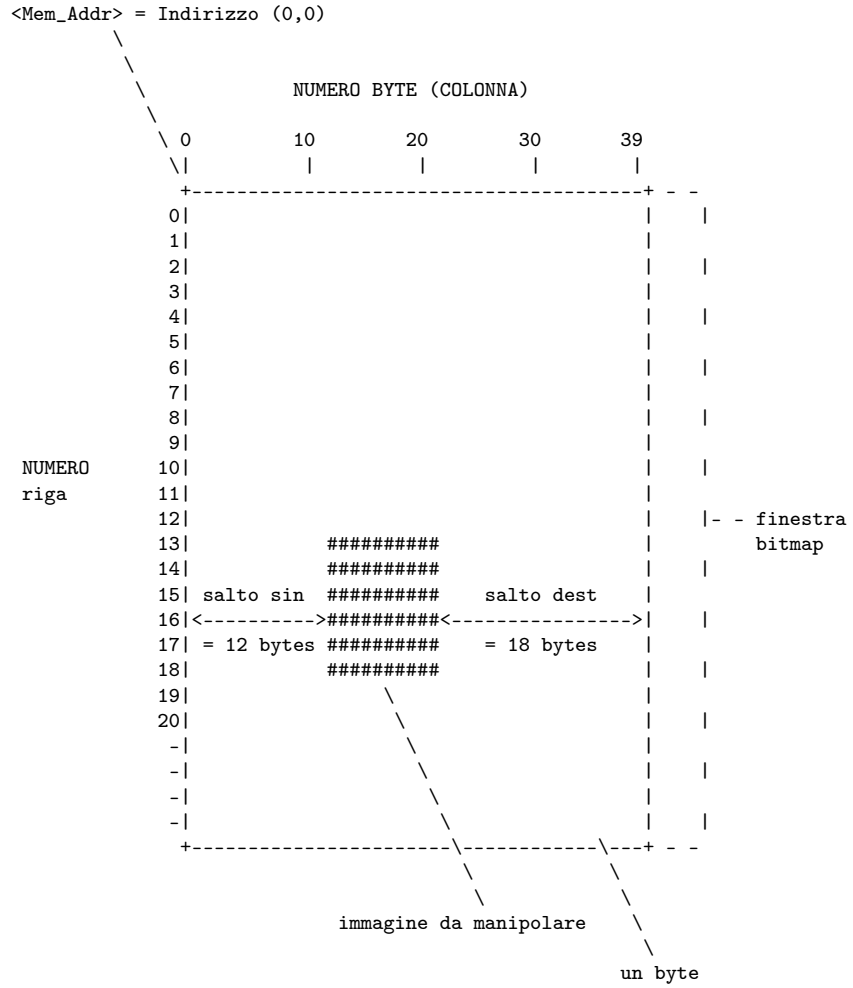
È possibile assegnare un modulo diverso per ogni canale DMA. In questo modo i dati possono essere copiati e spostati fra bitplanes di differenti larghezze. Il valore del modulo viene scritto in 4 appositi registri, uno per ogni canale DMA: `BLTAMOD` per il canale A (*$dff064*), `BLTBMOD` per il B (*$dff062*), `BLTCMOD` per il C (*$dff060*), `BLTDMOD` per il D (*$dff066*).I valori del modulo sono in byte, non words. Siccome il blitter può operare solo su words, il bit meno significativo è ignorato, questo significa che il valore del modulo deve essere pari. Il valore, positivo o negativo, viene aggiunto automaticamente ai registri che puntano agli indirizzi (`BLTxPT`) ogni volta che il blitter ha finito di copiare una riga, in modo da calcolare l'indirizzo della prima word della riga successiva. Valori negativi del modulo possono essere utili in molti casi, ad esempio per ripetere una linea settando il modulo come la larghezza del bitplane al negativo. Abbiamo già visto nella lezione 5 come replicare una linea mettendo il modulo copper `BPL1MOD/BPL2MOD` a -40, o comunque -LunghezzaLinea.

A questo punto sappiamo come copiare un rettangolo all'interno di una bitmap. Riassumiamo con un esempio tutti i calcoli necessari.

Supponiamo di voler operare su una sezione di un bitmap 320x200, che inizia alla riga 13, word 6 (dove entrambi sono numerati da zero) larga 5 word. Per prima cosa dobbiamo calcolare l'indirizzo della prima word del rettangolo, e poi scriverlo nel registro `BLTxPT` del canale che ci interessa. Il calcolo viene fatto nel modo seguente: prendiamo l'indirizzo del prima word del bitplane, aggiungiamo 13*20*2 bytes per calcolare l'indirizzo del primo byte della riga 13 (infatti ogni riga occupa 20 words=40 bytes) e infine aggiungiamo 12 byte (=6 words) per arrivare alla giusta posizione orizzontale. La larghezza è di 5 words (10 byte). Alla fine di ogni riga, dobbiamo saltare 30 byte per arrivare all'inizio della riga seguente, quindi usiamo un

modulo di 30. In generale, la larghezza (in words) raddoppiata più il valore modulo (in byte) dovrebbe equivalere alla piena larghezza, in byte, del bitplane che contiene l'immagine.

Questi calcoli sono illustrati nella figura che mostra i valori richiesti usati nei registri blitter BLTxMOD e BLTxPTR (BLTxPTH e BLTxPTL).




```
BLTxPTR = <Mem_Addr> + (40*13) + 12
        = <Mem_Addr> + 532

BLTxMOD = 12 + 18
        = 30 bytes
```

Fig. 7 calcoli per BLTxPTR e BLTxMOD

A questo punto è bene fare una sosta e guardarsi un pò di esempi.

In `lezione9c1.s` e `lezione9c2.s` trovate semplici esempi di copia di aree rettangolari. Studiateli attentamente, concentrandovi sul calcolo degli indirizzi e dei moduli usati nelle blittate.

In `lezione9c3.s` c'è un esempio nel quale viene effettuata una blittata con modulo negativo.

In `lezione9d1.s` e `lezione9d2.s` vedrete i primi esempi di animazione con il blitter.

L'idea è molto semplice, per dare l'idea del movimento basta disegnare la nostra figura ogni volta in una posizione diversa, un pò come si faceva con gli sprites. Diversamente da allora, però,

prima di disegnare la figura nella nuova posizione, dovremo cancellarla dalla vecchia, altrimenti otterremmo un effetto "scia". In questi 2 esempi spostiamo di volta in volta la figura in basso di una riga, aggiungendo ogni volta 40 bytes all'indirizzo `BLTxPT`.

In `lezione9d3.s` applichiamo la stessa tecnica per spostare la figura in orizzontale. Notate però che modificare l'indirizzo equivale a spostare il rettangolo a destra (o a sinistra) di una o più word. Poiché una word corrisponde a 16 pixel, in questo modo possiamo spostare orizzontalmente la figura solo a scatti di 16 pixel, il che rende, come potete vedere nell'esempio il movimento poco fluido, e troppo veloce.

Finora ci siamo limitati a disegnare figure con il pixel più a sinistra in una posizione multipla di 16. Per avere un movimento fluido, invece, è necessario poter disegnare la figura in una posizione arbitraria dello schermo. Facciamo un esempio, immaginando di avere l'immagine di un auto che vogliamo spostare sullo schermo. Calcolando opportunamente l'indirizzo del rettangolo che la contiene, possiamo "blittare" la nostra auto a partire da una qualunque delle word che formano lo schermo. Se la nostra immagine di auto, per esempio, ha lo sportello a 5 pixel dalla sua estrema sinistra, potremo spostarlo, assieme con la macchina, a 5 pixel dall'inizio di una qualche word dello schermo. Se vogliamo spostarla verso destra, possiamo "blittarla" a partire dalla word successiva. Il risultato sarebbe uno "scatto" di 16 pixel ogni volta. Ma se vogliamo far scorrere quell'auto a destra o a sinistra di un pixel alla volta, o comunque vogliamo blittarla in una posizione orizzontale che non sia multipla di 16, come si fa?

Dobbiamo fare in modo che i pixel che formano l'immagine vengano copiati NON a partire dal primo bit della prima word, a partire da un bit arbitrario all'interno di tale word, come illustrato dalla seguente figura.

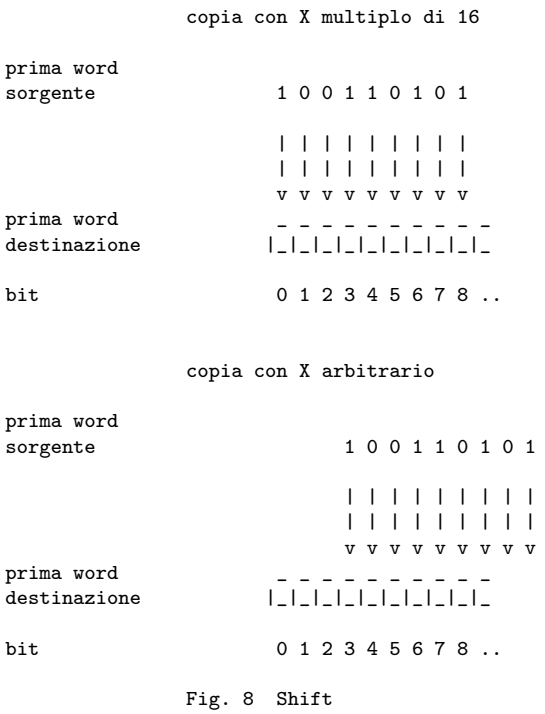


Fig. 8 Shift

In pratica dobbiamo shiftare i bit che compongono la figura da destra verso sinistra. Il blitter possiede degli shifter hardware per i canali A e B, che shiftano a destra tutti i bit delle word che vengono lette dai canali A e B. I bit vengono shiftati di numero di posizioni che può variare da 0

a 15. Shiftare di 0 posizioni equivale a non shiftare affatto: tutte le blittate che abbiamo visto (e fatto) finora erano blittate con shift di 0 posizioni. Il valore di shift per il canale A è assegnato con i bit dal 15 al 12 del registro BLTCON0 (*$dff040*); il valore di shift del canale B è assegnato con i bit dal 15 al 12 di BLTCON1 (*$dff042*). Se vi ricordate finora avevamo sempre lasciato questi bit al valore 0, che indica appunto uno shift di 0 posizioni. Il canale C invece è un proletario, non ha shifter. (Per chi se lo è dimenticato, shiftare dei bit significa "scorrere" dei bit verso destra o verso sinistra...) L'operazione di shifting viene eseguita contemporaneamente alla normale copia e non influenza la velocità del blitter: qualunque sia il valore di shift, il tempo impiegato per la blittata è sempre lo stesso.

Grazie allo shift, possiamo disegnare una figura avente il pixel più a sinistra in una posizione X arbitraria. Infatti, calcolando come di consueto l'indirizzo della destinazione, possiamo disegnare la figura ad una posizione X multipla di 16. Attivando contemporaneamente lo shifter, possiamo spostarla ulteriormente verso destra per farle raggiungere la posizione desiderata. Per esempio, supponiamo che si voglia una posizione X di 38 pixel. Mediante il calcolo dell'indirizzo possiamo spostare la figura 32 pixel (32 è multiplo di 16) a destra del bordo, 0 e possiamo spostarci a destra di altri 6 bits (38-32=6) impostando uno shift di 6.

In generale se X non è multiplo di 16, facendo la divisione intera X/16 otteniamo un risultato intero (che usiamo per calcolare l'indirizzo destinazione) e un resto che ci dice di quanto deve essere lo shift. (Ricordo che la divisione intera è una divisione nella quale non vengono calcolate le cifre decimali del risultato e viene ottenuto un resto, come si fa in prima elementare; per esempio 7/3=2 col resto di 1). Nel caso di una posizione orizzontale X=100, abbiamo 100/16=6 con il resto di 4 (infatti 16*6=96 e 100-96=4); dunque la distanza tra la prima word destinazione e la prima word della riga è pari a 6 words, ossia 12 bytes, e il valore di shift è di 4 bit.

Prima di iniziare ad usare lo shift dobbiamo però capire bene come funziona. Per cominciare, alcuni bit naturalmente sono shiftati a destra fuori delle word a cui appartenevano. Da sinistra deve essere shiftato dentro qualcosa per rimpiazzare i bit usciti. Cosa in particolare? Per la prima word della blittata, sono shiftati dentro degli zeri; per ogni successiva word della stessa blittata, i bit shiftati dentro una word sono quelli shiftati fuori dalla word precedente. Insomma, quello che esce da una parte (la destra) rientra dall'altra (sinistra!) nella word seguente. Facciamo un piccolo esempio, aiutandoci con una figura per capirlo meglio. Supponiamo di copiare 3 words (possono formare un rettangolo alto una riga e largo 3 word, oppure alto 3 righe larghe 1 word, non fa differenza dal punto di vista dello shift), applicando un valore di shift pari a 3. Guardiamo cosa succede:

```
SORGENTE
 word 1                  word 2                  word 3
1000110001010101        0001001001000110        1010101010101010

DESTINAZIONE
 word 1                  word 2                  word 3
0001000110001010        1010001001001000        1101010101010101
^^^                     ^^^                     ^^^
questi 3 bit sono gli   questi sono i 3 bit     questi sono i 3 bit
zeri shiftati dentro    shiftati fuori dalla    shiftati fuori dalla
la prima word           prima word e rientrati  word 2 e rientrati
                        nella seconda word      nella word 3
```

Fig. 9 shift

Notate che gli ultimi 3 bit della word 3 della sorgente **non** vengono copiati da **nessuna parte**!

Ad esempio, consideriamo una blittata larga tre words e alta due words, con uno shift di 4 bit. Per semplicità, assumiamo che sia una copia normale da A a D. La prima word che sarà scritta

in D è la prima word presa da A, shiftata a destra di quattro bit con 4 bit azzerati shiftati dentro dalla sinistra. La seconda word sarà la seconda word presa da A, shiftata a destra, con i quattro bit meno significanti (a destra) della prima word shiftati dentro. Dopo, scriverà la prima word della seconda riga presa da A, shiftata quattro bit, con i quattro bit meno significativi dell'ultima word dalla prima riga shiftati dentro. Questo continuerà finché la blittata non è terminata.

In `lezione9e1.s` potete vedere un esempio di uso dello shift, che permette ad una figura di muoversi di un pixel alla volta verso destra. Il risultato però non è molto buono a causa del fatto che i bit che vengono shiftati fuori da una word, vengono shiftati dentro la word successiva, che si trova una riga più in basso. Quindi i bit che escono a destra rientrano da sinistra nella riga successiva! La situazione è illustrata dalla seguente figura, assumendo uno shift di 4 bit:

```
SORGENTE
word 1          1000001111100000
  "  2          1100111111111000
  "  3          1111111111101100
  "  4          1111111111111110
  "  5          1100111111111000
word 6          1000001111100000


DESTINAZIONE
word 1          0000100000111110
  "  2          0000110011111111
  "  3          1000111111111110
  "  4          1100111111111111
  "  5          1111100111111111
word 6          1000100000111110
                ^^^^
                queste 4 colonne di bit sono costituite dai bit entrati da
                sinistra: come vedete (tranne che per la prima riga) in ogni
                riga entrano i bit usciti dalla riga precedente.

                Fig. 10  Shift di un rettangolo
```

Fortunatamente questo problema si risolve in modo molto semplice. Se ci pensate bene, quello che noi vorremmo, è che i bit che escono a destra da una word, rientrino da sinistra **non** nella riga successiva, ma piuttosto **nella word più a destra!** Dobbiamo quindi "coinvolgere" nella blittata anche le word più a destra. Ciò si può fare semplicemente aumentando la larghezza della figura aggiungendo a destra una "colonna" di word *di valore zero*. In questo modo, la colonna in più è invisibile, ed inoltre i bit shiftati fuori dalle word che la compongono saranno tutti zeri e perciò non daranno fastidio rientrando nelle word della riga seguente. Per chiarirvi le idee ecco cosa succede:

```
SORGENTE
                word 1          word 2
riga 1          10000011111000000000000000000000
  "  2          11001111111110000000000000000000
  "  3          11111111111011000000000000000000
  "  4          11111111111111100000000000000000
  "  5          11001111111110000000000000000000
  "  6          10000011111000000000000000000000
                           ^^^^^^^^^^^^^^^^
                           Questa è la colonna di words aggiunta

DESTINAZIONE
                word 1          word 2
riga 1          00001000001111100000000000000000
```

```
  "  2          00001100111111111000000000000000
  "  3          00001111111111101100000000000000
  "  4          00001111111111111110000000000000
  "  5          00001100111111111000000000000000
  "  6          00001000001111100000000000000000
                ^^^^             ^^^^
                |                Questi 4 bit sono usciti dalla word 1
                |                e sono entrati nella word 2
                |
                Questi 4 bit sono usciti dalla word 2 della riga precedente e
                entrati nella word 1 (tranne quelli entrati nella word 1 della
                riga 1, che sono azzerati automaticamente)

                Fig. 11  Shift di un rettangolo
```

Nell'esempio `lezione9e2.s` vedrete applicata questa tecnica, che consente di spostare una figura verso destra di un numero di pixel compreso tra 1 e 15 (infatti i valori di shift possibili vanno da 0 a 15 compresi). Nell'esempio `lezione9e3.s` finalmente vediamo la nostra figura spostarsi verso destra un numero arbitrario di pixel. In pratica vengono combinati insieme gli esempi `lezione9d3.s` e `lezione9e2.s`

9.4 Blittate "a colori"

Fino a questo momento ci siamo limitati a considerare immagini formate da un solo bitplane, cioè a 2 soli colori. Normalmente, quando si lavora con immagini a più colori, si dispongono i bit-plane consecutivamente in memoria, in modo che subito dopo l'ultima word di un bit-plane ci sia la prima word del bit-plane successivo. L'immagine risulta quindi strutturata nel modo seguente:

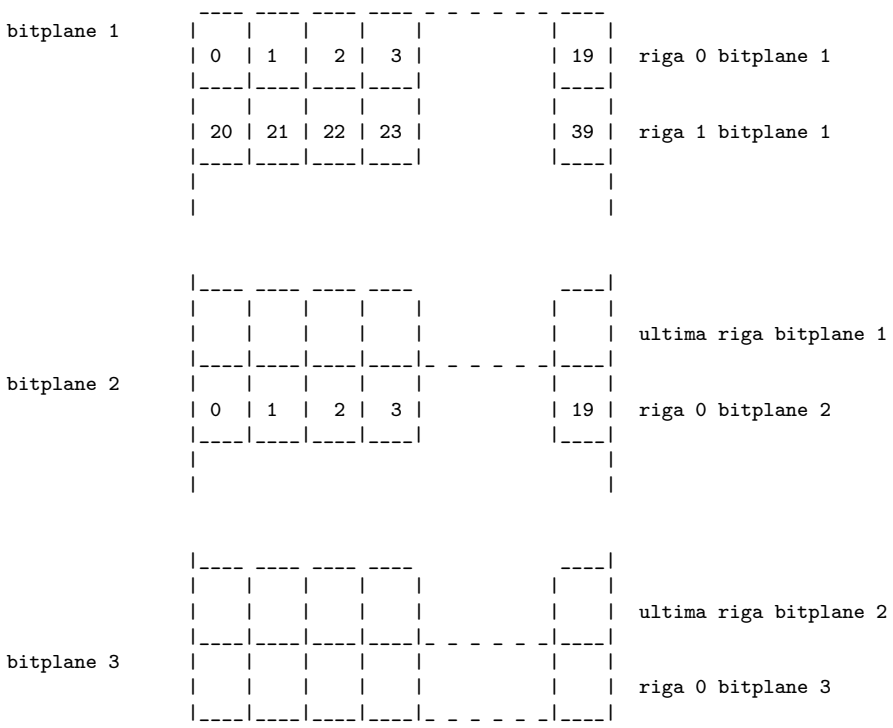
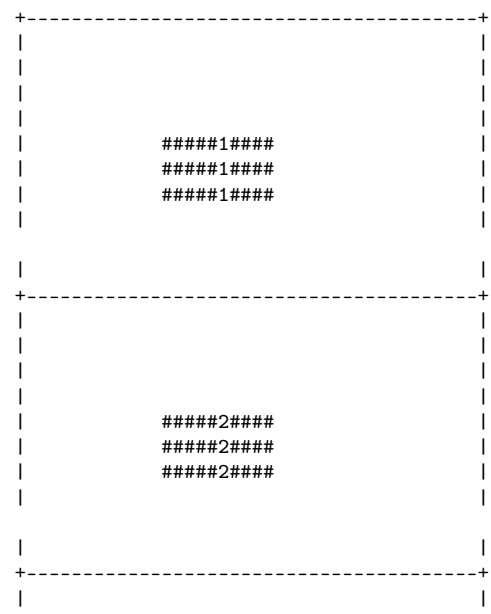


Fig. 12 Rappresentazione in memoria di un'immagine
a più bitplane (ogni quadrato è una word)

Come già sapete, un bit-plane largo H words e alto V righe, occupa H*V words, ovvero 2*H*V bytes (normalmente H=20 e V=256, quindi un bit-plane occupa 40*256 bytes). Questo significa che, poiché i bit-plane sono disposti in memoria uno di seguito all'altro, se il bit-plane 1 inizia all'indirizzo *PLANE1*, il bit-plane 2 inizierà all'indirizzo *PLANE2*=PLANE1+2*H*V. Analogamente il bit-plane 3 inizia all'indirizzo *PLANE3*=PLANE2+2*H*V e così via. La stessa formula vale per determinare l'indirizzo di una word del secondo bit-plane conoscendo l'indirizzo della word corrispondente del primo bit-plane: per esempio la settima word del primo bit-plane ha indirizzo *INDIRIZZO1* = PLANE1+2*7, mentre la settima word del secondo bit-plane ha indirizzo *INDIRIZZO2* = PLANE2+2*7 = PLANE1+2*H*V+2*7. Ma siccome PLANE1+2*7 = *INDIRIZZO1*, abbiamo la seguente formula:

```
INDIRIZZO2 = INDIRIZZO1+2*H*V.
```

Questa formula ci sarà molto utile tra poco. Un immagine rettangolare contenuta in uno schermo con N bitplane, sarà costituita da N rettangoli, uno per bitplane. Quindi, per manipolarla con il blitter, basta eseguire una blittata per ogni bit-plane. Nella figura sottostante potete vedere uno schermo con 3 bitplane, con evidenziata un'immagine alta 3 righe. In memoria, le righe di ogni bitplane, costituiscono un diverso rettangolo di words (abbiamo indicato in ogni riga dell'immagine il bitplane a cui essa appartiene). Come vedete le righe di ogni bitplane sono vicine tra loro e distanti dalle righe degli altri planes, pertanto devono essere manipolate con blittate diverse.



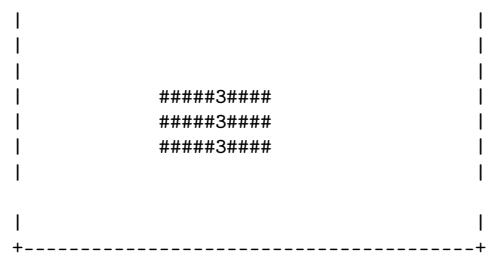


Fig. 13 Schermo con evidenziata un'immagine.

Per esempio, se abbiamo una figura da disegnare sullo schermo, prima blittiamo il primo plane della figura nel primo plane dello schermo, poi il secondo plane della figura nel secondo plane dello schermo, poi facciamo lo stesso con il terzo plane e via via con gli altri. Di solito quindi si fa un loop di blittate, tipo il seguente:

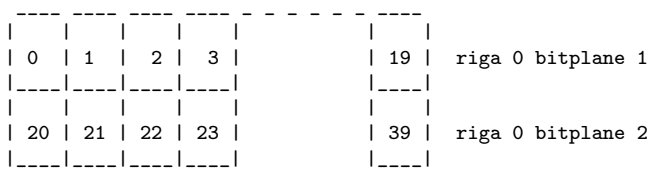
```
1  move.w  #NUMEROPLANES-1,d1      ; contatore del loop
2  LOOP:
3  waitblit:                       ; aspetta che il blitter abbia finito
4  btst    #6,2(a5)        ; la blittata precedente
5  bne.s   waitblit
6
7  move.l  #$09f00000,$40(a5)      ; bltcon0 e BLTCON1 - copia da A a D
8
9  ;       carica gli altri registri del blitter
10
11 ;       avvia la blittata
12
13 dbra    d1,LOOP         ; fai il loop
```

I valori da caricare nei registri del blitter sono sempre gli stessi ad ogni blittata, tranne ovviamente per quanto riguarda i registri BLTxPT, perché gli indirizzi dei vari bit-plane sono diversi. A questo punto entra in gioco la formula che abbiamo visto. Mediante tale formula infatti, conoscendo gli indirizzi da scrivere nei registri BLTxPT per la prima blittata (cioè per la blittata relativa al primo bit-plane), siamo in grado di calcolare gli indirizzi da scrivere nei registri BLTxPT per le blittate successive (cioè relative ai bit-plane successivi). È sufficiente mettere in una variabile l'indirizzo relativo al primo bit-plane, e di aggiungere a tale indirizzo 2*H*V ad ogni loop.

Nell'esempio `lezione9f1.s` potete vedere questa tecnica applicata. Non sempre comunque si usano loop di questo tipo.

Negli esempi `lezione9f2.s` e `lezione9f3.s` ci sono altri esempi di blittate "a colori".

Esiste però un'altro modo di disporre in memoria i bitplane, che ci consente di blittare in un colpo solo tutti i bitplanes di un'immagine, chiamato *Interleaved Bitmap* ovvero "bitmap" interlacciata. Come suggerito dal nome, questa tecnica consiste nel "mischiare" tra loro le righe dei vari planes. Invece di mettere prima tutte le righe del primo plane, poi quelle del secondo e così via, mettiamo prima la riga 0 (la prima) del primo bitplane, poi la riga 0 del secondo bitplane e poi in ordine le righe 0 degli altri planes; dopo le righe 0 di tutti i planes, mettiamo la riga 1 del primo plane, poi la riga 1 del secondo, e poi tutte le righe 1 degli altri planes; poi continuiamo così con le altre righe. Per capirlo bene guardate la figura seguente e confrontatela con la figura 12 dove è illustrata la disposizione normale dei bit-planes.



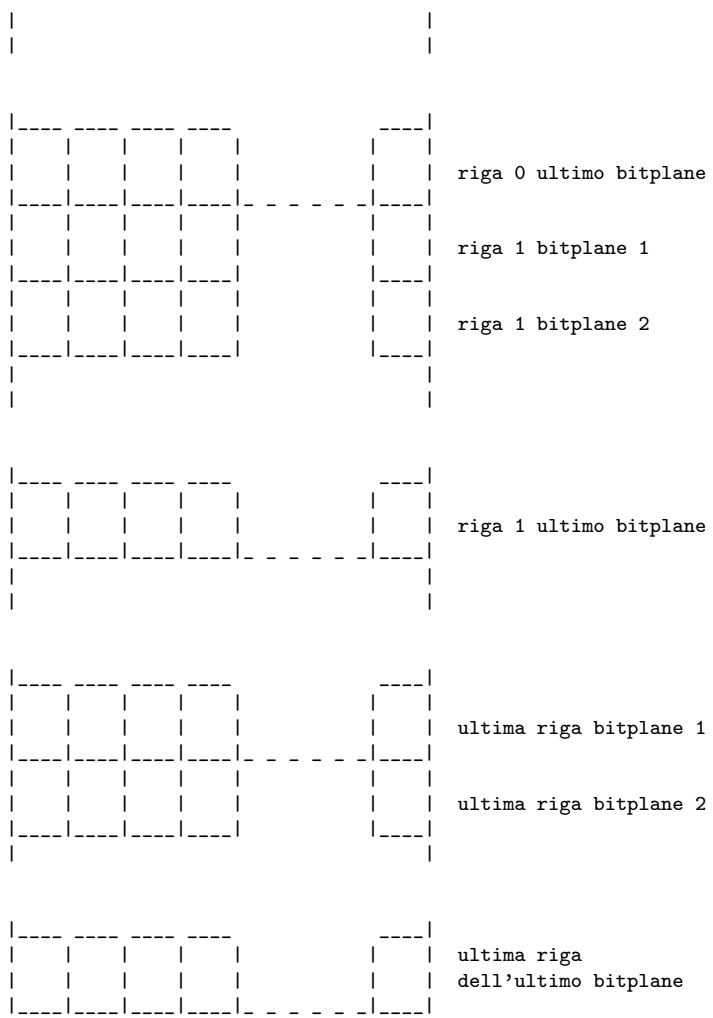


Fig. 14 Rappresentazione in memoria di un'immagine a più bitplane (ogni quadrato è una word) con la tecnica INTERLEAVED (o RAWBLIT).

Innanzitutto vediamo come si possono visualizzare immagini in questo formato, lasciando da parte un attimo il blitter. La quantità di words che compongono le righe è sempre la stessa. Quello che cambia è la disposizione relativa delle righe. Ciò per noi comporta 2 modifiche alla procedura che usiamo di solito per visualizzare i bitplanes. La prima riguarda il modo in cui calcoliamo gli indirizzi da mettere nei registri BPLxPT. Normalmente, per puntare i bitplane nella copper list, calcoliamo gli indirizzi dei bitplane successivi al primo, a partire dall'indirizzo del primo, aggiungendo ad esso ogni volta il numero di bytes occupati da una riga, moltiplicato per il numero di righe che formano il bitplane. Questo perché la prima riga di un bitplane è memorizzata dopo l'ultima del bit-plane precedente, e quindi "dista" dalla prima riga del bit-plane precedente un numero di righe pari all'altezza del bitplane stesso. Con la disposizione interleaved, invece, la riga o di un bitplane è memorizzata subito dopo la riga 0 del bitplane che lo precede.

Questo vuol dire che nel loop che calcola gli indirizzi dei bitplane dovremo aggiungere ogni volta all'indirizzo di un bitplane semplicemente il numero di bytes occupati da UNA riga, per ottenere l'indirizzo del bitplane seguente. Dobbiamo osservare inoltre che, a differenza del caso normale, le righe che formano un bitplane NON sono disposte consecutivamente in memoria. Infatti, tra la riga Y e la riga Y+1 ci sono le righe degli altri bitplane. Ciò significa che il puntatore al bitplane, ogni volta che arriva alla fine di una riga, deve "saltare" le righe degli altri bitplanes, per andare a puntare l'inizio della prossima riga.

Come avrete già intuito, per farlo saltare dobbiamo utilizzare il modulo. Vi ricordo infatti che anche i bitplanes hanno i loro moduli, contenuti nei registri `BPLxMOD` (dove x=1 per i bitplanes dispari e x=2 per i pari). Con la disposizione normale dei bitplanes, siccome subito dopo la fine di una riga inizia la riga successiva, mettiamo il modulo a 0 (a meno che non vogliamo fare l'effetto flood o abbiamo un'immagine più grande dello schermo). Vediamo invece il valore da mettere con la disposizione interleaved. Indichiamo con N il numero di bitplane che usiamo. Consideriamo il bitplane 1: all'inizio della riga Y il registro `BPLPT1` punta alla prima word della riga Y del bitplane 1. Mentre la riga Y viene visualizzata sul monitor, il registro `BPLPT1` si sposta puntando le words seguenti. Alla fine della riga Y, `BPLPT1` punta alla prima word della riga Y del bitplane2. A questo punto gli viene sommato il modulo. Noi vogliamo che `BPLPT1` vada a puntare la prima word della riga Y+1 del bitplane 1. Dobbiamo quindi far saltare al puntatore le righe 2, 3, ecc... fino a N. In totale si tratta di N-1 righe (per esempio se abbiamo 4 bitplane, dobbiamo saltare la riga Y dei bitplanes 2, 3 e 4, cioè 3 righe). Quindi se una riga occupa L words, ovvero 2*L bytes, il valore corretto del modulo è 2*L*(N-1).

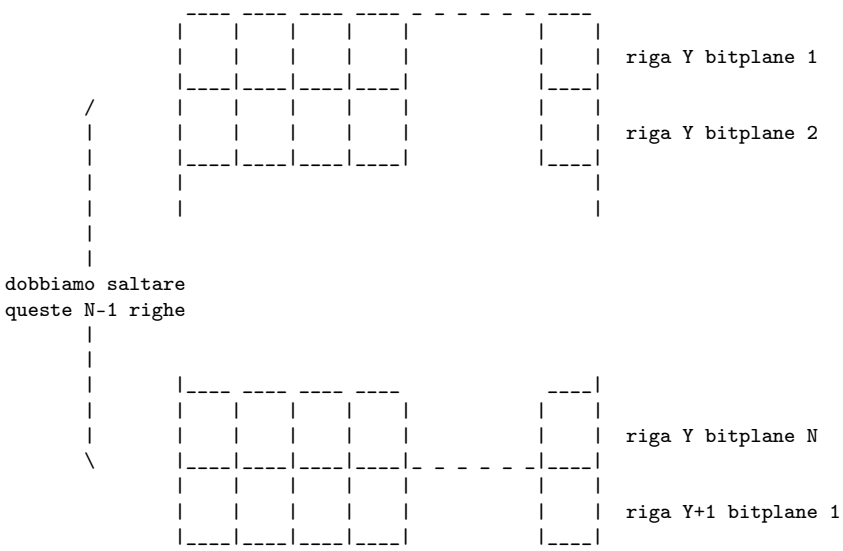


Fig. 15 Valore del modulo con la tecnica INTERLEAVED.

Naturalmente tutte le immagni che voliamo visualizzare sullo schermo dovranno avere i bitplane disposti nel formato interleaved. Se un immagine è definita direttamente nel nostro sorgente (tramite delle `DC.w ...`) , dobbiamo disporre le righe come previsto dal formato. Se invece vogliamo tenere l'immagine in un file esterno da includere con la direttiva `INCBIN`, dobbiamo convertirla **non** in formato RAW (che è il formato normale) ma in formato interleaved. Tutti i programmi di conversione supportano questo formato, anche se molti lo chiamano con altri nomi. In particolare il KEFRENS CONVERTER che abbiamo usato nel corso, chiama questo formato "RAW-BLIT". Altri converter lo chiamano "RASTER MODULO". Fate quindi attenzione a

convertire l'immagine nel giusto formato, altrimenti non vedrete nulla e passerete ore a cercare nel vostro programma un BUG inesistente !

In `lezione9g1.s` vedete un esempio di visualizzazione di una bitmap interleaved.

Vediamo ora perché questo formato è conveniente nell'uso del blitter. Nella figura seguente, viene mostrato uno schermo interleaved con evidenziata al suo interno un'area rettangolare. Come potete vedere, le righe che formano i vari bitplanes sono "mischiate" tra loro, e formano un unico rettangolo in memoria (abbiamo indicato in ogni riga dell'immagine il bitplane a cui essa appartiene). Confrontate questa figura con la figura 13 che mostrava una situazione analoga in uno schermo "normale". Nel caso normale, le righe degli N bitplanes dell'immagine, formano N distinti rettangoli di words, ognuno alto tante righe quante sono le righe dell'immagine. Nel caso interleaved invece, le righe degli N bitplanes, mischiandosi, formano un unico rettangolo di word. Notate che questo rettangolo ha altezza pari all'altezza dell'immagine moltiplicata per il numero di bitplanes che la formano. Nella figura abbiamo infatti un'immagine di 3 bitplanes alta 3 linee. Il rettangolo di words ha 9 righe.

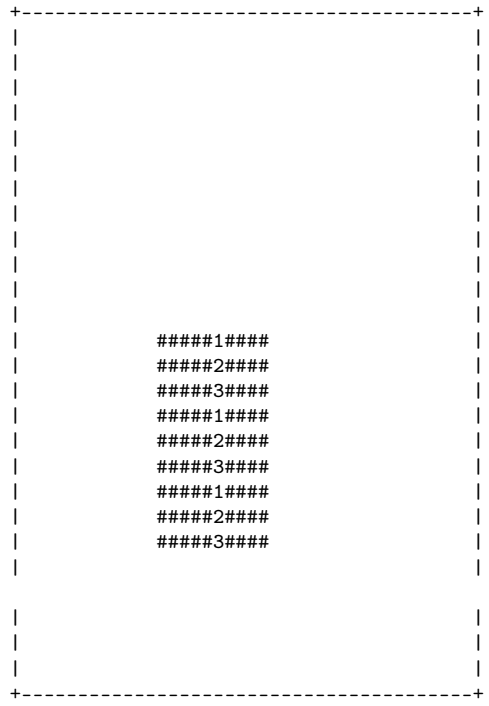


Fig. 16 Schermo INTERLEAVED con evidenziata un'immagine.

Il fatto che nel formato interleaved le righe dei bitplanes di un'immagine formino un unico rettangolo in memoria, è molto importante perché ci consente di operare sull'immagine mediante una sola blittata. Naturalmente, questa blittata è diversa dalle blittate che facciamo nel caso normale. Innanzitutto, è diversa la dimensione della blittata. Infatti nel caso normale ogni blittata ha altezza pari all'altezza dell'immagine, mentre nel caso interleaved il rettangolo di words ha un'altezza pari all'altezza dell'immagine moltiplicata per il numero di bitplanes che la formano, e quindi tale deve essere l'altezza della nostra blittata.

In secondo luogo, è diverso il modo in cui calcoliamo gli indirizzi delle blittate, in particolare dobbiamo cambiare il modo di calcolare l'indirizzo della prima word di una riga. Nel caso normale, abbiamo visto che se il rettangolo da blittare inizia alla riga Y, la "distanza" (offset)

della prima word della riga Y dall'inizio del bitplane è pari a Y*(NUMERO DI BYTES OCCUPATI DA UNA RIGA). E questo è logico, perché in uno schermo normale le righe di un bitplane sono consecutive in memoria. In uno schermo INTERLEAVED, invece, le cose sono diverse perché le righe di un bitplane non sono consecutive. Infatti, come sapete, dopo la riga Y del primo bitplane, ci sono le righe Y degli altri bitplane, e dopo di esse la riga Y+1 del primo bitplane. Quindi, la distanza tra la prima word della riga Y del primo bitplane e la prima word della riga Y+1 del primo bitplane, è uguale al numero di bytes occupati dalle righe Y di tutti i bitplane della figura. Con lo stesso ragionamento capite facilmente che la distanza tra la prima word della riga Y del primo bitplane e l'inizio dello schermo è pari a:

```
Y*(NUMERO_DI_BYTES_OCCUPATI_DA_UNA_RIGA)*(NUMERO_DI_PLANES)
```

In conclusione, quindi, il calcolo dell'indirizzo per blittare un rettangolo che inizia alle coordinate X e Y per uno schermo INTERLEAVED diventa:

```
Indirizzo_word = (Indirizzo_bitplane)+N*2

con:
N=(Y*(NUMERO_WORD_CHE_FORMANO_UNA_RIGA)*(NUMERO_PLANES))+(X/16).
```

Fare una sola blittata invece che tante, oltre a rendere più semplice il programma, lo rende anche più veloce. Si badi bene che il tempo impiegato dal blitter è (più o meno) lo stesso, in quanto è vero che facciamo una sola blittata, ma essa ha altezza pari alla somma delle altezze delle blittate del caso normale, e quindi richiede lo stesso tempo, perché la velocità del blitter è determinata in sostanza dal numero di words che esso deve manipolare, e cioè dalla dimensione della blittata. Fare una sola blittata, però, avvantaggia notevolmente il processore, come potete capire dal seguente schema, che confronta le operazioni da effettuare nei 2 casi (schermo formato da 3 bitplanes):

```
        SCHERMO NORMALE                     SCHERMO INTERLEAVED

1)      attendi la fine                     attendi la fine
        della (eventuale) blittata          della (eventuale) blittata
        precedente                          precedente

2)      carica i registri del               carica i registri del
        blitter per la prima                blitter per la prima
        blittata                            e unica blittata

3)      attendi la fine
        della prima blittata

4)      carica i registri del
        blitter per la seconda
        blittata

5)      attendi la fine
        della seconda blittata

6)      carica i registri del
        blitter per la terza
        blittata
```

Come vedete, nel caso di schermo interleaved, il processore deve fare meno operazioni, e sopratutto deve attendere una sola volta che il blitter finisca, mentre nel caso di schermo normale deve attendere un numero di volte pari al numero dei bitplanes. Poiché durante un attesa il

processore non fa nulla di utile e non ha bisogno di riposo, è opportuno farlo lavorare il più possibile diminuendo il numero di attese.

L'esempio `lezione9g2.s` è la versione INTERLEAVED dell'esempio `lezione9f1.s`. Guardateli insieme, notando le differenze che essi presentano.

L'esempio `lezione9g3.s`, invece, è la versione INTERLEAVED dell'esempio `lezione9f3.s`. Confrontate anche questi.

9.5 Maschere

Il blitter ha la possibilità di mascherare la prima e l'ultima word di ogni riga che passa attraverso il canale A. Mascherare vuol dire leggere solo alcuni bit di tali word e ignorare gli altri. Questa operazione viene effettuata grazie a due registri, che finora avevamo usato senza spiegarne il significato. Questi due registri sono chiamati `BLTAFWM` (*$dff044*) e `BLTALWM` (*$dff046*), e servono rispettivamente per mascherare la prima e l'ultima word di ogni riga letta attraverso il canale A. Ognuno di essi contiene una word, detta maschera. Il blitter quando legge la prima o l'ultima word di una riga esegue un'operazione logica di AND tra la word letta e la maschera corrispondente. I bit della word letta dal canale A in corrispondenza dei quali c'è un bit settato a 0 nella maschera verranno cancellati. Vediamo qualche esempio:

```
word letta dal
canale A        %1001101100010111

maschera        %1111111100000000
---------------------------------

risultato       %1001101100000000
```

in questo modo abbiamo selezionato solo gli 8 bit più a destra della word.

```
word letta dal
canale A        %1001101100010111

maschera        %1111110000111111
---------------------------------

risultato       %1001100000010111
```

in questo modo abbiamo azzerato i 4 bit al centro della maschera. Se azzeriamo completamente la maschera, cancelliamo tutta la word:

```
word letta dal
canale A        %1001101100010111

maschera        %0000000000000000
---------------------------------

risultato       %0000000000000000
```

Se invece poniamo la maschera al valore $ffff=%1111111111111111=-1 la maschera non cancella nulla, ovvero "fa passare" tutta la word:

```
word letta dal
canale A        %1001101100010111

maschera        %1111111111111111
```

```
--------------------------------

risultato       %1001101100010111
```

In tutti gli esempi che abbiamo visto finora non abbiamo avuto bisogno di mascherare nulla e infatti abbiamo inizializzato entrambe le maschere al valore $ffff.

La prima word di ogni riga (cioè la word più a sinistra) è "ANDizzata" con `BLTAFWM`, e l'ultima word (la word più a destra) è "ANDizzata" con `BLTALWM`. Potete facilmente ricordarlo perché la F nel nome `BLTAFWM` indica "First" che come tutti sanno significa "prima" e la L in `BLTALWM` indica "Last", cioè ultima. Naturalmente le 2 maschere possono essere diverse tra loro (sennò a che ci servirebbero 2 registri?). Se la larghezza della riga è una singola word, entrambe le maschere vengono applicate simultaneamente alla stessa word. Poiché i 2 registri `BLTAFWM` e `BLTALWM` hanno indirizzi consecutivi è possibile inizializzarli con una sola istruzione `MOVE.L #maschera,$dff044`. È importante notare che le maschere vengono applicate ai dati **prima** di eseguire lo SHIFT. I canali B e C non hanno invece la possibilità di mascherare le words lette.

Nell'esempio `lezione9h1.s` mostriamo l'effetto delle maschere con semplici operazioni di copia.

In `lezione9h2.s` abbiamo una dimostrazione dell'utilità delle maschere nell'estrarre da un'immagine solo la parte che ci interessa.

In `lezione9h3.s` e `lezione9h4.s` presentiamo 2 nuovi effetti realizzati con l'ausilio delle maschere.

Gli esempi `lezione9h2r.s`, `lezione9h3r.s` e `lezione9h4r.s` sono le versioni in formato rawblit (interleaved) di `lezione9h1.s` `lezione9h2.s` e `lezione9h3.s`. Fate un confronto incrociato, notando tutte le differenze che ci sono (in particolare notate che tutte le routines in versione interleaved non hanno la necessità di fare un loop per blittare su ogni plane, e pertanto hanno una struttura molto più semplice).

Dopo aver visto i nuovi effetti, torniamo ad occuparci di uno vecchio, cioè del pesce che nuota sullo schermo, per scoprire che, con le nostre nuove conoscenze sul blitter, possiamo realizzare un'importante miglioria. Abbiamo visto, infatti che per shiftare correttamente una figura, è necessario aggiungere a destra della figura una "colonna" di word azzerate. Questo fatto ci costringe a sprecare più memoria del necessario per memorizzare le figure. Ma ora, grazie alle maschere, possiamo evitare questo spreco. Per shiftare è necessario che l'ultima word di ogni riga della figura sia azzerata.

Invece di leggere direttamente dalla memoria una word azzerata, possiamo leggere una word di qualsiasi valore e azzerarla tramite le maschera. Siccome il mascheramento viene effettuato **prima** dello shift, al circuito di shift arriverà comunque l'ultima word di ogni riga azzerata, e tutto si svolgerà come se la word azzerata fosse stata letta dalla memoria. Visto che non ha importanza il valore dell'ultima word della riga, possiamo leggere una word di qualsiasi valore.

Proviamo allora a fare il seguente giochino: non aggiungiamo nessuna word a destra dell'immagine, ma senza dirlo al blitter, cioè settiamo la larghezza della blittata come se ci fosse una word in più a destra della figura. Il blitter, quindi, dopo aver letto l'ultima word di una riga, penserà di dover leggere ancora una word, e pertanto leggerà la word successiva a l'ultima della riga. Che cos'è questa word? Se usiamo un immagine in formato normale, sarà la prima word della riga successiva dello stesso bitplane, mentre se l'immagine è in formato interleaved sarà la prima word di una riga di un altro bitplane. In ogni caso sarà comunque una word non nulla, ma per noi non c'è problema perché la possiamo azzerare con la maschera. A questo punto abbiamo solo un problemino: siccome abbiamo letto una word di troppo, il puntatore della sorgente si è spostato in avanti di una word, pertanto quando inizierà a leggere la prossima riga partirà dalla seconda word invece che dalla prima. Come si può far tornare indietro il puntatore? Naturalmente con il vecchio trucco del modulo negativo! Settando il modulo della sorgente a -2 (il

modulo si esprime in bytes) il blitter si riposiziona sulla prima word della riga seguente. Riassumiamo tutto tornando all'esempio del pesce che abbiamo usato per illustrare lo shift. Abbiamo dunque un'immagine di un solo bitplane, larga 1 word e alta 6 righe. Come abbiamo detto, NON aggiungiamo la colonna di word sulla destra.

```
SORGENTE
                word 1
riga 1          1000001111100000
  "  2          1100111111111000
  "  3          1111111111101100
  "  4          1111111111111110
  "  5          1100111111111000
  "  6          1000001111100000

                Fig. 17 NON aggiungiamo nessuna colonna di words
```

Tuttavia, facciamo finta che la colonna in più ci sia, e quindi blittiamo un rettangolo largo 2 words e alto 6 righe. Il blitter legge quindi 2 words per ogni riga, prendendo come seconda word la prima word della riga successiva. Vediamo in particolare, con l'aiuto della figura seguente, cosa accade durante la lettura della prima riga:

```
SORGENTE
                word 1
riga 1          1000001111100000--------
  "  2          1100111111111000--------+------------------------
  "  3          1111111111101100        |                        |
  "  4          1111111111111110        |                        |
  "  5          1100111111111000        |                        |
  "  6          1000001111100000        |                        |
                                         |                        |
                                         V                        V
WORDS LETTE
DAL CANALE A                    1000001111100000        1100111111111000
                                         |                        |
                                         |                        |
                                         V                        V
L'ULTIMA WORD DELLA
RIGA VIENE MASCHERATA           1000001111100000        0000000000000000
                                         |                        |
                                         |                        |
                                         V                        V

SHIFT (2 pixel)                 0010000011111000        0000000000000000
                                         |                        |
                                         |                        |
                                         V                        V

                                Scritta al canale D     Scritta al canale D

                Fig. 18  Shift con azzeramento dell'ultima word.
```

Come vedete la seconda word letta viene azzerata prima di essere shiftata. Dopo lo shift le 2 word vengono scritte attraverso il canale D. Nel frattempo il puntatore al canale A si è spostato in avanti di 2 words, e punta cioè alla prima word della terza riga. Noi invece dobbiamo farlo puntare alla prima word della seconda riga, cioè dobbiamo farlo tornare indietro di una word. Usiamo quindi un modulo pari a -2. Gli spostamenti del puntatore sono illustrati dalla figura seguente:

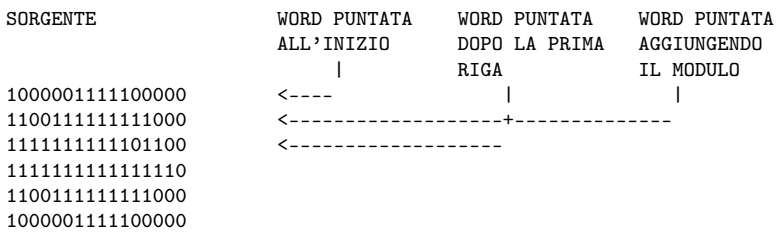


Fig. 19 Movimento del puntatore alla sorgente.

Per vedere il nostro pesce in azione consultate l'esempio `lezione9i1.s`.

Ormai sappiamo muovere molto bene delle figure sullo schermo usando il blitter. Queste figure vengono chiamate *BOB* che è un abbreviazione del termine inglese *Blitter OBject*, ovvero oggetti creati dal blitter. Con i BOB Possiamo fare le stesse cose che sappiamo fare con gli sprites hardware. I BOB sono più lenti degli sprites, perché il blitter impiega comunque un certo tempo per copiare dati. Per contro, però i BOBS non soffrono delle limitazioni degli sprites riguardo a dimensione, colori e numero massimo. Infatti un BOB può essere grande quanto vogliamo noi (è ovvio però che al crescere delle dimensioni cresce la quantità di memoria occupata, e di conseguenza il tempo necessario al blitter per spostarlo), e può avere un numero di colori pari a quello dello schermo. Inoltre non c'è nessun limite per quanto riguarda il numero di bob contemporaneamente sullo schermo (ovviamente però, più bob ci sono, più tempo perdiamo per disegnarli). "Che bello", direte voi, "possiamo iniziare a fare un gioco!". Un attimo, non esaltiamoci troppo. Siamo proprio sicuri di saper fare con i BOB le stesse cose che possiamo fare con gli sprites?

Guardiamo `lezione9i2.s` e il suo "gemello" in formato interleaved `lezione9i2r.s`.

Abbiamo un BOB colorato che spostiamo liberamente con il mouse sullo schermo. Però c'è un problema... muovendo il BOB cancelliamo lo sfondo! Questo con gli sprite non accade, in quanto gli sprite sono dei piccoli bitplanes separati dai bitplanes dello sfondo. I BOB invece vengono disegnati proprio sui bitplane dell'immagine di sfondo, quindi in parte la sovrascrivono.

Una prima soluzione al problema la presentiamo negli esempi `lezione9i3.s` e `lezione9i3r.s` (naturalmente il secondo è la versione rawblit del primo).

Come vedrete, però non è ancora soddisfacente.

Nell'esempio `lezione9i4.s` proviamo un'altra soluzione, ma anch'essa presenta problemi.

Nell'esempio `lezione9i5.s`, invece vediamo un esempio di bob mosso dal joystick che esce parzialmente dallo schermo.

Abbiamo iniziato a conoscere i BOB, ma per ora non abbiamo raggiunto un risultato soddisfacente, cioè riuscire a fare con i BOB le operazioni tipiche videogiochi a causa del problema dello sfondo. Purtroppo con quello che sappiamo finora non si può fare di meglio.

Ma non preoccupatevi: ci sono ancora molte cose da imparare sul blitter, e una di queste ci aiuterà a risolvere il problema! Forza e coraggio dunque, la strada è ancora lunga!

9.6 Copia di zone di memoria sovrapposte

Illustreremo ora un'altra caratteristica del blitter prendendo spunto dalla copia di rettangoli, operazione che ormai conosciamo bene. Cosa succede se la sorgente e la destinazione della blittata sono sovrapposte, ovvero sono 2 rettangoli di word che hanno delle parti in comune? E' ovvio che la blittata modificherà tutta la destinazione, comprese quindi le parti in comune con la sorgente. La copia tra zone sovrapposte consiste quindi nel mettere il contenuto della sorgente PRIMA della copia nella destinazione. Dopo la copia, il contenuto della sorgente sarà cambiato.

Pertanto, dopo la copia, la destinazione NON sarà uguale alla sorgente!! Piuttosto, ripetiamo, essa sarà uguale a come era la sorgente PRIMA della copia!

Insomma immaginate che la destinazione sia una fotografia scattata alla sorgente, e che durante il tempo impiegato dal fotografo per sviluppare la foto, la sorgente sia invecchiata rapidamente tanto da apparire molto diversa da come appare nella foto. È sempre effettuare una copia in tali condizioni? Dobbiamo studiare bene il problema. Vediamo cosa succede con un esempio di una copia di un rettangolo alto 2 righe e largo 3 words. Supponiamo che la sorgente si trovi più in basso della destinazione, come illustrato dalla figura seguente:

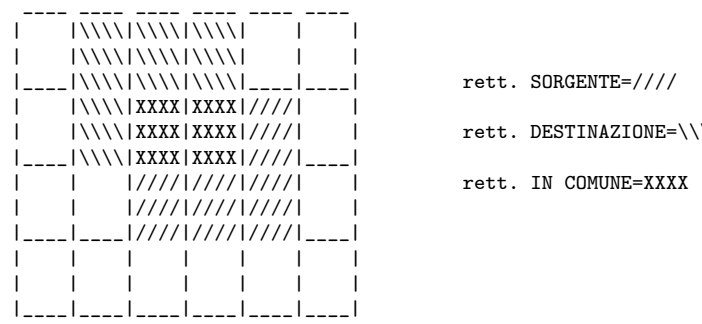


Fig. 20 Blittata tra rettangoli sovrapposti

Analizziamo, con l'aiuto di una serie di figure, le successive fasi dell'operazione. Indichiamo con le lettere A, B, C, D, E, F il contenuto delle 6 words che vogliamo copiare, e con il simbolo "?" il contenuto delle word che non ci interessano, e che quindi possiamo anche cancellare. Prima di iniziare la copia, abbiamo questa situazione:

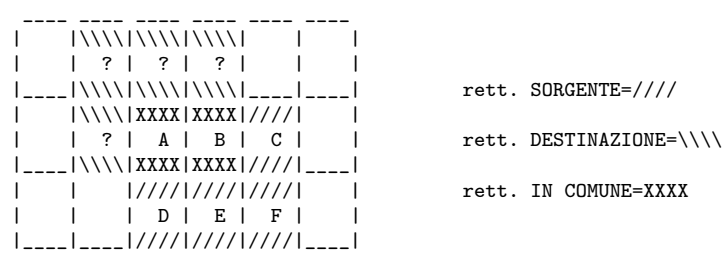


Fig. 21a Blittata tra rettangoli sovrapposti

Come sappiamo il blitter copia le words una alla volta partendo da quella più in alto a sinistra e proseguendo verso il basso e verso destra. La prima riga viene letta e copiata su una zona della destinazione non in comune, e che quindi possiamo sovrascrivere tranquillamente. Ecco la situazione dopo la copia della prima riga:

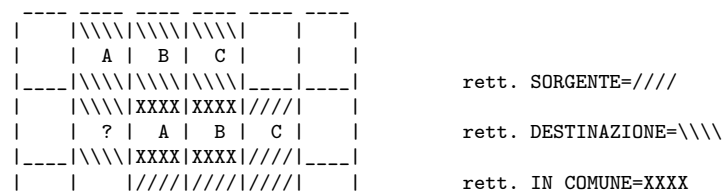


Fig. 21b Blittata tra rettangoli sovrapposti

A questo punto dobbiamo copiare la seconda riga. La seconda riga della destinazione si sovrappone con la prima riga della sorgente. Questo significa che quando scriveremo i dati nella destinazione, sovrascriveremo una parte della sorgente, distruggendone il contenuto. Osservate però che i dati sovrascritti appartengono alla PRIMA riga della sorgente, che noi abbiamo già copiato, e che quindi non ci serve più. Pertanto non ci sono problemi. La situazione dopo la copia della seconda (e ultima) riga è la seguente:

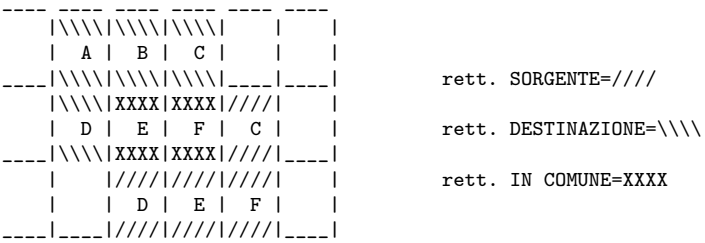


Fig. 21c Blittata tra rettangoli sovrapposti

Abbiamo ottenuto proprio ciò che volevamo, in quanto ora il rettangolo destinazione è la copia esatta del contenuto del rettangolo sorgente PRIMA che iniziassimo la blittata. Notate che ora, invece il contenuto della sorgente è cambiato, ma ciò era inevitabile.

Potete vedere tutto ciò in pratica nell'esempio `lezione9l1.s`.

Sembrerebbe dunque che la sovrapposizione tra sorgente e destinazione non crei problemi. Proviamo però ad esaminare il caso in cui la destinazione si trovi più in basso della sorgente:

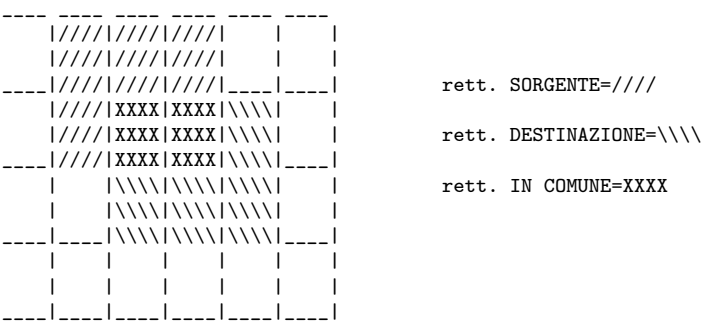


Fig. 22 Blittata tra rettangoli sovrapposti

Prima della blittata, la situazione è la seguente:

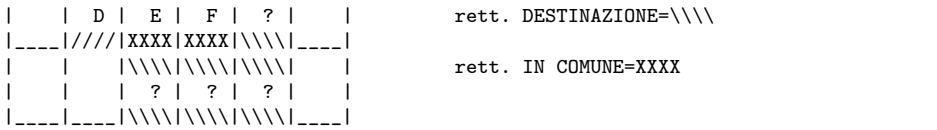


Fig. 23a Blittata tra rettangoli sovrapposti

Iniziamo con il copiare la prima riga. La prima riga della destinazione è parzialmente sovrapposta con la seconda riga della sorgente, che non è ancora stata copiata. Ecco quello che si ottiene:

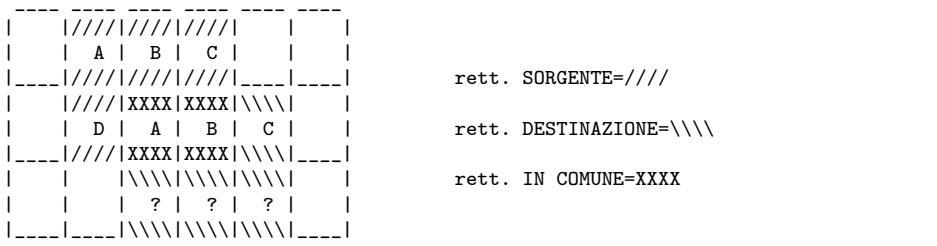


Fig. 23b Blittata tra rettangoli sovrapposti

Come potete notare ci siamo persi i valori E ed F! Sembra proprio che stavolta la copia non riuscirà bene! Comunque copiamo anche la seconda riga, e vediamo che succede.

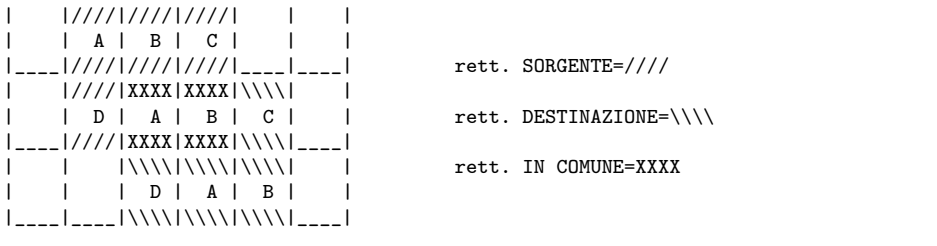


Fig. 23c Blittata tra rettangoli sovrapposti

Ecco fatto. La blittata è finita ma il risultato non è quello che volevamo. Siete convinti?

No? Allora, guardatevi l'esempio `lezione9l2.s` e convincetevene!

Cerchiamo di capire perché la prima volta ha funzionato e stavolta no. Il problema nasce quando scriviamo sulle parti della destinazione che si sovrappongono con la sorgente, perché così facendo sovrascriviamo alcuni dati. Nel primo caso non ci sono stati problemi perché i dati sovrascritti li avevamo già copiati. Ciò è accaduto perché la sorgente si trova più in basso (ad indirizzi maggiori) della destinazione, e la sovrapposizione si verifica tra la prima riga della sorgente e la seconda riga della destinazione. Siccome il blitter copia partendo dalla prima riga, i dati della prima riga della sorgente vengono copiati PRIMA di essere sovrascritti dalla seconda riga della destinazione.

Nel secondo caso, invece, la sorgente si trova più in alto (ad indirizzi minori) della destinazione, e la sovrapposizione si verifica tra la seconda riga della sorgente e la prima della destinazione. I dati della seconda riga della sorgente, quindi, vengono sovrascritti durante la copia della prima riga, e cioè PRIMA di essere copiati a loro volta, pertanto vengono persi. Per risolvere questo problema, bisognerebbe copiare prima la seconda riga e poi la prima.

Ciò è possibile utilizzando il *modo discendente* del blitter. Quando si utilizza questo modo, il blitter esegue la copia (o una qualunque altra operazione) in senso inverso a quanto fa di solito, cioè parte dalla word più in basso a destra del rettangolo e prosegue verso sinistra e verso l'alto. Le words che blitta seguendo questo percorso hanno indirizzo via via minore. Si dice perciò che il blitter *discende* lungo la memoria, da cui il nome del modo di funzionamento (per contrasto, il modo normale viene detto anche MODO ASCENDENTE, infatti normalmente vengono blittate words con indirizzi via via crescenti). Prima di esaminare nel dettaglio come si usa il blitter in modo discendente, ritorniamo al problema della copia di regioni sovrapposte e verifichiamo che il modo discendente è la giusta soluzione. La situazione di partenza è la seguente:

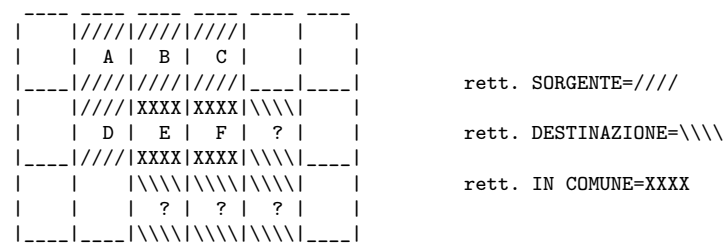


Fig. 24a Blittata tra rettangoli sovrapposti

Questa volta usiamo il modo discendente, perciò iniziamo a copiare a partire dall'ultima riga. In questo modo all'inizio non scriviamo sulla parte sovrapposta:

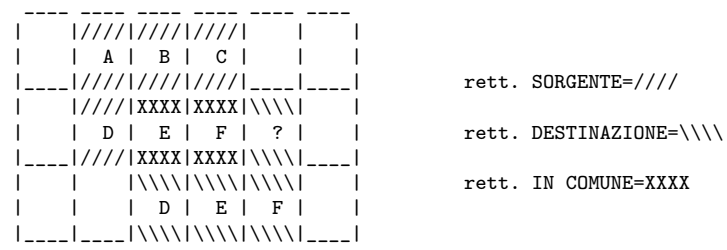


Fig. 24b Blittata tra rettangoli sovrapposti

Adesso copiamo la prima riga. Nel farlo sovrascriviamo la seconda riga della sorgente, ma poiché l'abbiamo già copiata non è un problema:

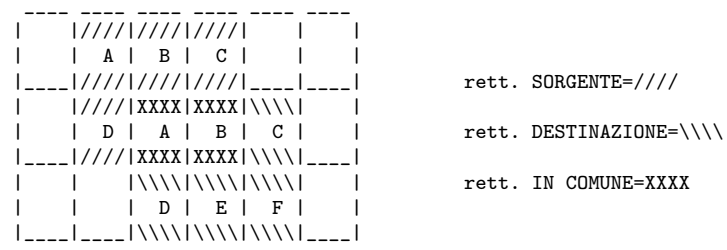


Fig. 24c Blittata tra rettangoli sovrapposti

OK! Questa volta ci siamo. Ora la destinazione ha lo stesso aspetto della sorgente prima di effettuare la blittata. Per concludere, possiamo quindi dire che quando effettuiamo una copia con sorgente e destinazione sovrapposte, se la sorgente si trova ad indirizzi di memoria maggiori

della destinazione, si deve fare la blittata in modo normale (*ascendente*), se invece la sorgente si trova ad indirizzi di memoria minori, si deve utilizzare il modo *discendente*.

A questo punto possiamo scendere nel dettaglio del modo discendente. Innanzitutto, il modo discendente va attivato mediante un bit di controllo. Si tratta del bit 1 del registro `BLTCON1`, che se settato ad 1 attiva il modo discendente, mentre quando viene azzerato (come abbiamo fatto finora) attiva il modo ascendente. Come abbiamo già detto, in modo discendente il blitter va "all'indietro" cioè si sposta tra locazioni di memoria di indirizzo via via minore. Per questo è necessario che i puntatori dei canali DMA puntino all'inizio della blittata alla word della blittata che ha indirizzo maggiore di tutte, cioè la prima word che verrà blittata. Si tratta, come sapete, della word più in basso e più a destra del rettangolo di words che verrà blittato. Per esempio, nel caso in cui si voglia blittare un rettangolo largo 3 words e alto 2 righe, si dovranno inizializzare i puntatori con l'indirizzo della terza word della seconda riga del rettangolo, che nella figura è indicata con due asterischi (**)

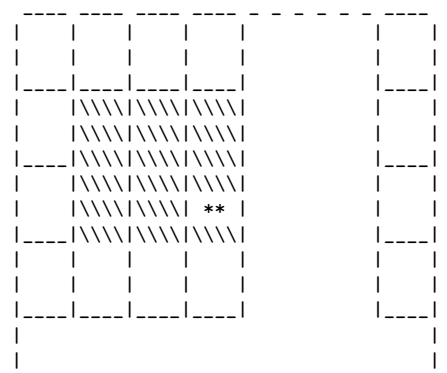


Fig. 25 Rettangolo di word con evidenziata la word
da puntare all'inizio della blittata

Per calcolare l'indirizzo di tale word si segue un ragionamento simile a quello fatto nel caso ascendente. Dobbiamo calcolare la distanza (offset) di tale word dall'inizio del bitplane. Supponiamo di conoscere le coordinate Xa e Ya del pixel più in alto a sinistra del rettangolo, ed anche la larghezza in words L e l'altezza A del rettangolo. La word che ci interessa appartiene all'ultima riga del rettangolo che ha coordinata Yb=Ya+A. L'offset della prima word di tale riga è dato dalla seguente formula:

```
OFFSET_Y = 2*(Yb*NUMERO_WORDS_PER_RIGA)                 nel caso normale e

OFFSET_Y = 2*(Yb*NUMERO_WORDS_PER_RIGA*NUMERO_PLANES)   nel caso interleaved.
```

Ora dobbiamo calcolare la distanza tra la prima word della riga e l'ultima word del rettangolo. Come sappiamo tale distanza è data da 2*(Xa/16). D'altronde tra la prima e l'ultima word del rettangolo ci sono L-1 words, che equivalgono ad una distanza (che si esprime in bytes) di 2*(L-1). Sommando le 2 differenze abbiamo:

```
OFFSET_X=2*(Xa/16+L-1).
```

```
\____________________/\_______________________/
        |                        |
    Xa/16 words            L words

distanza tra word A e word B = 2*(Xa/16)
distanza tra word B e word C = 2*(L-1)
```

Fig. 26 Calcolo OFFSET_X

Quindi l'indirizzo da scrivere nei puntatori dei canali DMA è dato da:

```
INDIRIZZO_WORD = INDIRIZZO_BITPLANE+OFFSET_Y+OFFSET_X.
```

Per quanto riguarda i moduli e la dimensione della blittata, non ci sono differenze rispetto al caso ascendente, vengono calcolati tutti con le stesse formule. Ora possiamo finalmente copiare correttamente 2 regioni rettangolari sovrapposte anche quando la sorgente inizia ad un indirizzo di memoria minore di quello della destinazione: si tratta dell'esempio `lezione9l3.s`.

In modo discendente le maschere e lo shift si comportano diversamente rispetto al modo ascendente. Le maschere funzionano sempre nello stesso modo, ma cambiano le word a cui si applicano. La maschera contenuta in `BLTAFWM` viene applicata, come per il caso ascendente, alla prima word che blittiamo di ogni riga. Siccome però in modo discendente blittiamo al contrario, la prima word è la word più a destra del rettangolo, mentre in modo ascendente è la word più a sinistra. Allo stesso modo, la maschera contenuta in `BLTALWM` viene sempre applicata all'ultima word blittata di ogni riga, solo che in modo discendente tale word è la word più a sinistra. In sintesi:

- In modo ascendente (normale) `BLTAFWM` si applica alla word più a sinistra e `BLTALWM` alla word più a destra.
- In modo discendente `BLTAFWM` si applica alla word più a destra e `BLTALWM` alla word più a sinistra.

Se guardiamo l'immagine come appare sul video, passando al modo discendente le maschere si scambiano le colonne su cui operano. Per verificarlo caricate ed eseguite l'esempio `lezione9m1.s` che fa esattamente le stesse cose di `lezione9h1.s`, solo che opera in modo discendente. Vedrete che le maschere producono gli stessi effetti ma scambiando le colonne.

Lo shift, in modo discendente presenta una differenza fondamentale: viene fatto verso SINISTRA, anziché verso destra. Se specifichiamo un valore di shift pari per esempio a 2, la sorgente viene shiftata di 2 pixel **verso sinistra**. Utilizzando questa caratteristica possiamo realizzare l'effetto di scorrimento di un immagine verso sinistra. Lo trovate nell'esempio `lezione9m2.s`.

A questo punto, siamo finalmente in grado di realizzare uno degli effetti più classici delle demo: lo *ScrollText*, ovvero un testo che scorre sullo schermo da destra verso sinistra.

Un semplice ma significativo esempio è la `lezione9n1.s`, nella quale troverete tutte le spiegazioni. Mi raccomando studiate con particolare attenzione questo esempio perché saper fare uno scrolltext è assolutamente fondamentale per un demo-coder!

Nell'esempio `Lezione9n2.s` troverete lo scrolltext dell'intro del disco1.

Avete capito come funziona lo scrolltext? Se la risposta è affermativa, potete iniziare ad essere soddisfatti di quanto avete imparato. Ormai conoscete il funzionamento di base del blitter. Nella prossima lezione scopriremo i segreti più nascosti di questo potente amico, a partire dal più grande, il più difficile da capire, con il quale abbiamo avuto a che fare per tutta questa lezione ma che abbiamo sempre evitato: il funzionamento dei *MintTerms*!

CAPITOLO 10

LEZIONE 10 - BLITTER AVANZATO

In questa lezione apprenderemo l'uso delle caratteristiche più avanzate del blitter.

10.1 I MINTERMs

Nella lezione 9 abbiamo detto che il blitter ci permette di effettuare diversi tipi di operazioni. Abbiamo anche detto che il tipo di operazione è definito dai *MINTERMS*, che sono i bit da 0 a 7 del registro `BLTCON0`, ovvero il byte basso (detto byte `LF` - Logic Function) di tale registro. A seconda del valore che viene scritto in tali bit, cambia l'operazione realizzata dal blitter. Per esempio sappiamo che per cancellare la memoria il byte `LF` va settato al valore $00, mentre per copiare dal canale A al canale D al valore $f0. Questi valori non sono stati scelti a casaccio dai progettisti del blitter, ma seguono una logica ben precisa, che ora spiegheremo.

Innanzitutto precisiamo che le operazioni effettuabili dal blitter sono operazioni LOGICHE, ovvero NOT, AND e OR, che ormai dovreste conoscere bene (in realtà c'è anche chi riesce a farci operazioni aritmetiche, ma ne parleremo, forse, nel prossimo disco!). Il blitter inoltre può combinare diverse operazioni di questo tipo in una unica blittata. Ma andiamo con ordine. Come sapete il blitter ha 3 canali di ingresso e uno di uscita. Per il momento non preoccupiamoci della abilitazione o disabilitazione dei canali. Una blittata è un'operazione logica che prende 3 valori in ingresso attraverso i 3 canali A,B,C e produce un risultato attraverso il canale D. Come tutte le operazioni logiche, essa viene effettuata bit-a-bit, anche se il blitter legge (e scrive) sempre delle word, esattamente come fa il 68000 con un'istruzione logica tipo AND. Quindi ogni bit della word in uscita viene calcolato in base ai valori dei corrispondenti bit delle word in ingresso. I 3 bit in ingresso possono dar luogo a 8 diverse combinazioni. Un'operazione blitter viene definita stabilendo, per ogni possibile combinazione dei bit in ingresso, se il risultato in uscita sarà 0 oppure 1. In pratica, ad ognuno degli 8 minterms (bit da 0 a 7 di `BLTCON0`) viene associata una diversa combinazione dei bit in ingresso; se il minterm vale 0, vuol dire che la combinazione in ingresso produce come risultato 0, se invece vale 1, il risultato sarà 1.

Questo può essere visualizzato con una tavola di verità, come mostrato sotto. Sono listati i tre canali di sorgente, e i valori possibili per un singolo bit di ognuno. A fianco è riportato il bit associato ad ogni combinazione.

A	B	C	posizione BLTCON0
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Fig. 27 MINTERMS

Per esempio, se vogliamo che una blittata produca un'uscita pari a 1 quando l'ingresso A vale 0, il B vale 1 e il C vale 0, e che invece produca un uscita pari a 0 in tutti gli altri casi, dobbiamo settare a 1 il minterm 2, e azzerare tutti gli altri minterms. Quindi scriveremo il valore $04 nel byte LF. Per un altro esempio, il valore $80 (= 1000 0000 binario) in LF pone a 1 solo i bit della destinazione per i quali i bit corrispondenti delle sorgenti A, B, e C sono tutti settati a 1. Tutti gli altri bit della destinazione a cui corrispondono altre combinazioni per A, B, e C, sono azzerati. Questo perché i bit dal 6 allo 0 del byte LF assumono il valore 0. Naturalmente è possibile settare a 1 più di un minterm contemporaneamente. Per esempio se poniamo LF al valore $42 (= 0100 0010 in binario) "accendiamo" 2 minterms. Quindi con questo valore avremo un uscita pari a 1 in 2 casi: nel caso in cui A=0, B=0 e C=1 (corrispondente al bit 1 di LF) e nel caso A=1, B=1 e C=0 (corrispondente al bit 6 di LF). Negli altri casi avremo un uscita pari a 0. Cerchiamo ora di capire il significato dei valori dei minterms che abbiamo usato per la cancellazione e la copia. Nel caso della cancellazione si ha LF=$00.

Tutti i minterms valgono 0. Questo significa che per qualunque combinazione dei canali sorgente, viene prodotto in uscita sempre uno 0. In pratica qualsiasi cosa leggiamo, scriviamo sempre 0, cioè cancelliamo (In realtà durante la cancellazione non leggiamo nulla perché non abilitiamo i canali A, B e C, ma dobbiamo comunque mettere LF=$00, spiegheremo il perché in seguito). Per eseguire una copia da A a D, poniamo, come sapete, LF=$F0 (=%11110000). In questo modo l'uscita vale 1 in corrispondenza di 4 diverse combinazioni, mentre vale 0 nelle restanti 4. Come potete leggere nella tabella di fig.27, le combinazioni corrispondenti ai minterms che abbiamo settato a 1, sono tutte le combinazioni possibili con A=1, e allo stesso modo le combinazioni corrispondenti ai minterms settato a 0, sono quelle con A=0. Ciò significa che tutte le volte che A=1, l'uscita vale 1 e quando invece A=0 l'uscita vale 0, indipendentemente dal valore di B e di C. In pratica cioè l'uscita assume lo stesso valore del canale A, e quindi ne è la copia esatta. Se volessimo invece copiare dal canale B al canale D, dovremmo usare un diverso valore di LF, ponendo a 1 i minterms che corrispondono alle combinazioni con B=1 (che come si legge nella fig.27 sono i minterms 2,3,6 e 7) e azzerare gli altri (minterms 0, 1, 4 e 5), ottenendo LF=$CC (=%11001100). Programmando opportunamente i minterms si possono fare molte operazioni con il blitter.

Supponiamo per esempio di voler settare ad 1 tutti i pixel di un rettangolo (in pratica l'operazione inversa della cancellazione che invece setta tutti i bit a 0). Come per la cancellazione utilizziamo solo il canale di uscita. Quello che vogliamo è che l'uscita sia sempre 1, per qualun-

que combinazione degli ingressi. Per ottenere questo risultato poniamo ad 1 tutti i minterms, ottenendo LF=$FF.

Potete vedere questa operazione nell'esempio `lezione10a1.s`.

Nell'esempio `lezione10a2.s` mostriamo invece l'operazione NOT. Vi rimandiamo al listato per la spiegazione.

Passiamo ora ad un esempio di operazione a 2 operandi, per esempio l'OR. Vogliamo che l'uscita sia pari all'OR dei canali A e B. Ripensando alla tabella della verità dell'OR, si capisce che l'uscita deve valere 1 in tutti i casi in cui A=1 e in tutti i casi in cui B=1. Come potete vedere dalla fig. 27 in totale si tratta di 6 casi che danno luogo a LF=$FC.

L'esempio `lezione10b1.s` mostra appunto un operazione di OR, mentre l'esempio `lezione10b2.s` effettua un operazione di AND.

Un altro modo per calcolare il byte LF che realizza una particolare operazione è attraverso l'uso di diagrammi di Venn:

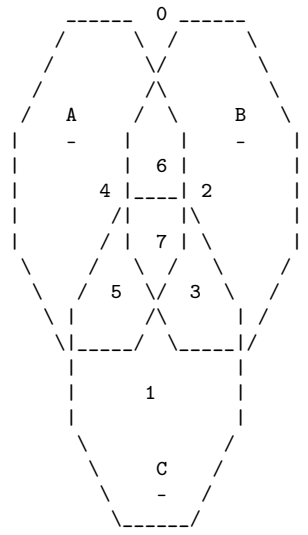


Fig. 28 Diagramma di Venn

Illustriamo l'uso di tale diagramma attraverso alcuni esempi

1. Per selezionare una funzione D=A (cioè destinazione = sorgente A solamente), selezionare solo i minterms che sono totalmente inclusi dal cerchio A nella figura sopra. Questa è la serie di minterms 7, 6, 5, e 4. Quando sono scritti come una serie di 1 per i minterm selezionati e di 0 per quelli non selezionati, il valore diventa:

```
Numero Minterm          7 6 5 4 3 2 1 0
Minterm selezionati     1 1 1 1 0 0 0 0
                        -----------------
                             F   0       ossia $F0
```

2. Per selezionare una funzione combinazione di due sorgenti, cercare i minterms da entrambi dei cerchi (la loro intersezione). Per esempio, la combinazione A "AND" B è rappresentato dall'area comune ai cerchi A e B, ossia i minterms 7 e 6.

```
Numero Minterm          7 6 5 4 3 2 1 0
Minterm selezionati     1 1 0 0 0 0 0 0
                        -----------------
                             C   0       ossia $C0
```

3. Per usare una funzione che è l'inverso, il "NOT", di uno dei sorgenti, es:

   ```
   NOT A
   ```

   prendere tutti i minterms non inclusi dal cerchio rappresentato da A. In questo caso, abbiamo i minterms 0, 1, 2, e 3.

   ```
   Numero Minterm          7 6 5 4 3 2 1 0
   Minterm selezionati     0 0 0 0 1 1 1 1
                           -----------------
                                0   F       ossia $0F
   ```

4. Per combinare minterms, cioè un "OR" tra essi, fare un OR dei valori. Per esempio, l'operazione (A AND B) OR (B AND C) diventa

   ```
   Numero Minterm                7 6 5 4 3 2 1 0
   A AND B                       1 1 0 0 0 0 0 0
   B AND C                       1 0 0 0 1 0 0 0
                                 -----------------
   (A AND B) OR (B AND C)        1 1 0 0 1 0 0 0
                                 -----------------
                                      C   8       ossia $C8
   ```

Ad ogni modo, se proprio volete risparmiarvi la fatica (male, SFATICATI! :), riportiamo una tavola dei valori Minterm più usati. In questa tabella viene usata una notazione diversa da quella usata finora:

- Se due termini sono adiacenti viene fatto un AND tra di loro (es. AB vuol dire A AND B);
- un trattino sopra ad un termine indica il NOT: (es. $\overline{A}$ vuol dire NOT A);
- se due termini sono separati da un $+$ viene fatto un OR tra di loro (es. A+B vuol dire A OR B);
- AND ha la precedenza più alta, così AB + BC è uguale a (A AND B) OR (B AND C). Ecco la tabella:

Operazione Selezionata	Valore LF	Operazione Selezionata	Valore LF
D = A	$F0	D = AB	$C0
D = $\overline{A}$	$0F	D = $\overline{AB}$	$30
D = B	$CC	D = $\overline{AB}$	$0C
D = $\overline{B}$	$33	D = $\overline{\overline{AB}}$	$03
D = C	$AA	D = BC	$88
D = $\overline{C}$	$55	D = $\overline{BC}$	$44
D = AC	$A0	D = $\overline{BC}$	$22

$D = \bar{A}C$	$50	$D = A\bar{C}$	$11
$D = \bar{A}C$	$0A	$D = A + \bar{B}$	$F3
$D = \bar{A}C$	$05	$D = \bar{A} + \bar{B}$	$3F
$D = A + B$	$FC	$D = A + \bar{C}$	$F5
$D = \bar{A} + B$	$CF	$D = \bar{A} + \bar{C}$	$5F
$D = A + C$	$FA	$D = \bar{B} + C$	$DD
$D = \bar{A} + C$	$AF	$D = \bar{B} + \bar{C}$	$77
$D = B + C$	$EE	$D = AB + \bar{A}C$	$CA
$D = \bar{B} + C$	$BB		

Fig. 29 Mintems più usati

NOTA: Per individuare il valore desiderato di LF per i vostri scopi potete usare anche l'utility "minterm", programmata da Deftronic, lo stesso del Trash'M'One. La breve utility in questione la trovate in questo disco. La sintassi è questa: per il NOT, si mette la lettera del canale non shiftata (lowercase), ad esempio "abc". Per il canale normale si usa la lettera shiftata (uppercase). Due lettere adiacenti significano un AND tra i canali, mentre se sono separate dal "+" significa un OR tra i canali.

```
                    __
Esempio: se si vuole ABC:

        minterm Abc

        risultato: $10

Esempio2: se si vuole solo la sorgente A:

        minterm A

        risultato: $F0  (come volevasi dimostrare)

Esempio3: se si vuole solo (A AND B) OR C:

        minterm AB+C

        risultato: $DA.
```

10.2 I BOBs

Siamo quasi arrivati al piatto forte della lezione, ovvero i BOB. Prima di affrontarli è necessario presentare un altra idea: il bit-plane maschera. Si tratta semplicemente di un bitplane che costituisce "l'ombra" di una figura, cioè un bitplane delle stesse dimensioni di una figura che ha settati ad 1 i pixel corrispondenti a pixel della figura, colorati con un colore diverso dallo sfondo, e invece settati a 0 i pixel che corrispondono al colore di sfondo della figura. Per esempio consideriamo la seguente tabella di numeri:

```
0020
0374
5633
0130
```

essa rappresenta un'immagine ad 8 colori (3 bitplanes) larga 4 pixel e alta 4 righe. Ogni numero indica il colore associato al pixel. La maschera di tale immagine è la seguente:

```
0010
0111
1111
0110
```

Osserviamo che i colori diversi dallo 0 (lo sfondo) hanno almeno un bitplane settato ad 1.

Pertanto la maschera può essere costruita a partire dalla figura facendo l'OR di tutti i bitplanes, come illustrato negli esempi `lezione10c1.s` e `lezione10c2.s` che vi permettono anche di ripassare l'utilizzo del blitter per fare operazioni logiche. In particolare, in `lezione10c2.s` mostriamo per la prima volta una blittata che usa tutti e 4 i canali del blitter.

Il Kefrens Converter, comunque, ha un'opzione per creare automaticamente la maschera di una figura. I bitplanes maschera sono utili perché ci permettono di visualizzare delle parti di un immagine, in base alla forma di un'altra immagine.

Ne vediamo esempi in `lezione10c3.s` e `lezione10c4.s`, dove utilizziamo una maschera a forma di cerchio per realizzare un riflettore che illumina un'immagine rendendone visibile una parte.

I 2 esempi, benché realizzino lo stesso effetto utilizzano tecniche molto diverse, come spiegato nei commenti. Studiate particolarmente bene `lezioneq4.s`, che è indispensabile per capire poi i BOB. In questo esempio, il bitplane maschera viene utilizzato per "selezionare" delle parti di un'immagine di 5 bitplanes. La selezione avviene effettuando una operazione di AND tra il bitplane maschera e i 5 bitplanes che costituiscono l'immagine. Poiché l'immagine è in formato normale, vengono effettuate 5 blittate distinte, una per ogni plane. La maschera, ovviamente è sempre la stessa per ogni blittata (è formata da un solo bitplane). Volendo applicare la tecnica dell'esempio `lezione10c4.s` ad uno schermo in formato interleaved, ci troviamo di fronte ad un problema. Quando operiamo in questo formato, infatti, blittiamo tutti i planes contemporaneamente. La maschera però ha la dimensione di un plane, e quindi non può essere usata in una blittata che ha una dimensione pari al numero di planes di cui si compone l'immagine. Per risolvere questo problema dobbiamo modificare la nostra maschera. Siccome ogni riga della maschera deve selezionare la riga corrispondente di **tutti** i bitplanes della figura, dobbiamo ripetere la riga tante volte quanti sono i bitplanes. In formato interleaved, quindi, dobbiamo usare un bitplane maschera che ha ogni riga ripetuta tante volte quanti sono i bitplane della figura. Nel caso della figura che abbiamo visto prima (3 planes) la nostra maschera interleaved è la seguente:

```
0010\
0010 |  - prima riga della maschera normale ripetuta 3 volte
0010/
0111
0111
0111
1111
1111
1111
0110
0110
0110
```

Come potete notare, poiché la figura ha 3 bitplanes, ogni riga della maschera in formato normale è stata ripetuta 3 volte per ottenere la maschera interleaved. Il formato interleaved, quindi, ci costringe ad utilizzare una maschera che occupa più memoria di quella richiesta dal formato normale.

L'esempio `lezione10c5.s` è la versione interleaved di `lezione10c4.s`, e ci permette di vedere in pratica quanto detto.

Se avete capito bene il funzionamento delle maschere, siete pronti per risolvere una volta per tutte il problema dello sfondo con i BOBS. Come sicuramente ricordate, nell'esempio `lezione9i3.s` siamo andati abbastanza vicini alla soluzione del problema. Lo sfondo viene salvato e successivamente ridisegnato al suo posto. L'unico problema è che nel rettangolo che racchiude la figura del BOB viene cancellato lo sfondo, e sostituito con il colore 0. In realtà quando disegniamo un BOB usiamo il colore 0 non come un colore qualsiasi ma semplicemente per denotare i pixel del rettangolo che non appartengono alla figura del BOB. E' esattamente la stessa cosa che facciamo con gli sprite, usiamo il colore 0 come "trasparente". Quando disegniamo il BOB sullo schermo vorremmo che al posto dei pixel colorati con il colore 0 apparisse lo sfondo, in pratica dovremmo poter scrivere sullo schermo solo i pixel di colore diverso da 0. Ciò non è possibile perché come sapete il blitter scrive (e legge) SEMPRE delle word INTERE.

Si adotta dunque una diversa strategia. Invece di fare una semplice copia del BOB sulla destinazione, facciamo una blittata più complicata. Leggiamo dalla memoria, oltre al BOB, anche lo sfondo, li "mischiamo" assieme, in modo che al posto dei pixel di colore 0 del BOB appaiano i pixel dello sfondo, e scriviamo il risultato sullo schermo. La strategia è illustrata nella figura seguente, nella quale abbiamo un BOB e un pezzo di sfondo di 6*8 pixel. Il simbolo "." rappresenta un pixel di colore 0, il simbolo "#" rappresenta un pixel del BOB di diverso colore, e il simbolo "o" rappresenta un pixel dello sfondo di diverso colore:

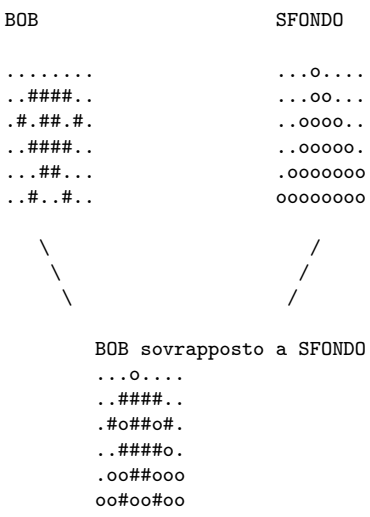


Fig. 30 Bob e sfondo

In questo modo otteniamo l'effetto desiderato. Resta da capire in che modo "mischiare" il BOB con lo sfondo. Per "mischiare" correttamente dobbiamo sapere quali pixel del BOB sono di colore 0 e quali no. Queste informazioni sono contenute nel bitplane maschera del BOB, che come sapete ha un bit a 0 per ogni pixel di colore 0 del BOB e un bit a 1 per ogni pixel di altro colore. L'operazione di mescolamento avviene dunque nel modo seguente:

- Per ogni pixel, leggiamo la maschera
- Se la maschera ha valore 1, copiamo il corrispondente pixel del BOB
- Se la maschera ha valore 0, copiamo il corrispondente pixel dello sfondo.

Possiamo realizzare questa procedura mediante una sola blittata, operando nel modo seguente: leggiamo la maschera attraverso il canale A del blitter, il BOB attraverso il canale B, lo sfondo attraverso il C, utilizziamo la maschera per selezionare i pixel da copiare (o dallo sfondo o dal BOB) e scriviamo il risultato nel canale D (l'assegnamento dei canali non è casuale). La selezione viene effettuata mediante la seguente equazione logica:

```
D = (A AND B) OR ( (NOT A) AND C)
```

Questa equazione si comporta esattamente come la procedura di selezione descritta in precedenza. Quando infatti la maschera A = 1 (cioè abbiamo un pixel del BOB di colore DIVERSO da 0) l'equazione si semplifica nel modo seguente:

```
D = (1 AND B) OR ( (NOT 1) AND C) = B OR (0 AND C) = B OR 0 = B
```

Quindi viene copiato il pixel del BOB. Quando invece A = 0 (cioè abbiamo un pixel del BOB di colore 0) l'equazione diventa:

```
D = (0 AND B) OR ( (NOT 0) AND C) = 0 OR (1 AND C) = 0 OR C = C
```

Quindi viene copiato il pixel dello sfondo. Questa equazione logica viene eseguita dal blitter (come potete calcolare voi stessi) ponendo LF = $CA, valore noto come *cookie cut* ovvero "taglio del biscotto". Come abbiamo accennato prima, l'assegnamento dei canali è stato fatto accuratamente sulla base delle caratteristiche dei canali stessi. Infatti per effettuare spostamenti fluidi orizzontali è necessario usare per il BOB e per la maschera lo shift del blitter; per questo il canale C (che non può fare lo shift) viene usato per lo sfondo. Inoltre, applichiamo il trucco di mascherare l'ultima word al bitplane maschera, in modo che l'ultima word di esso venga azzerata, facendo così che nell'ultima word venga blittato lo sfondo.

Gli esempi `lezione10d1.s` e `lezione10d1r.s` mostrano (rispettivamente in versione normale e interleaved) il tanto atteso BOB che si muove su uno sfondo.

10.3 La velocità del Blitter (e non solo)

È giunto il momento di occuparci di una questione molto importante: la velocità del blitter. Infatti come sapete, il blitter impiega una certa quantità di tempo per portare a termine i suoi compiti, ed è necessario tenere conto di ciò quando si programmano effetti complessi. Per misurare la velocità del blitter useremo una tecnica molto semplice, nota come "copper monitor", che ci mostra il risultato sullo schermo in tempo reale. La tecnica è semplicissima: utilizziamo un certo colore (di solito il nero) come sfondo. Poi, subito prima di iniziare la blittata cambiamo il colore di sfondo col processore, tramite un `MOVE.W #$xxx,$dff180`. Quando la blittata finisce rimettiamo lo sfondo al colore iniziale. In questo modo sappiamo che la blittata impiega un tempo proporzionale alla porzione di schermo colorata diversamente. Da notare che questa tecnica è usata per misurare qualsiasi tipo di routine, e in particolare è utilissima per capire quando diventa più veloce o più lenta a seguito di una modifica, ad esempio un'ottimizzazione. Un esempio è illustrato in `lezione10e1.s`.

In questo esempio usiamo il blitter per copiare un rettangolo sullo schermo. Sulla base di questo esempio possiamo iniziare a fare un pò di considerazioni sulla velocità del blitter. Innanzitutto, come avevamo già accennato, la velocità dipende dalle dimensioni della blittata. Provate nell'esempio a modificare l'altezza e/o la larghezza del rettangolo, e ve ne renderete conto da soli. Ciò è ragionevole, in quanto più è grande il rettangolo, maggiore è la quantità di words da spostare. Allo stesso modo, il numero di bitplanes influenza la velocità (provate in `lezione10e1.s` a cambiare il numero di iterazioni della routine "DisegnaOggetto"), in quanto più bitplanes ci sono e maggiore è la quantità di dati da spostare. L'esempio `lezione10e1r.s` è la versione rawblit dell'esempio precedente.

Eseguendolo noterete che esso risulta più veloce ma di pochissimo. Ma allora, vi chiederete, tutto il vantaggio del rawblit? In realtà, come abbiamo già detto, la tecnica rawblit è conveniente non tanto perché accelera il blitter, ma piuttosto perché fa risparmiare tempo al processore. Nei 2 esempi che abbiamo visto finora abbiamo misurato solo il tempo impiegato dal blitter.

Negli esempi `lezione10e2.s` e `lezione10e2r.s`, invece, usiamo diversi colori per evidenziare sia il tempo impiegato dal blitter che quello impiegato dal processore.

Il confronto tra questi esempi ci mostra appieno i vantaggi del modo rawblit: con questa tecnica il processore è impiegato pochissimo, giusto il tempo di caricare i registri del blitter, e poi è libero di eseguire altri compiti, a differenza di quanto accade con il modo normale, dove il processore deve attendere la fine di una blittata per lanciare la blittata del plane successivo. È chiaro che per sfruttare il vantaggio della tecnica rawblit è necessario che la routine successiva alla blittata NON impieghi il blitter. Infatti se (come accade negli esempi) dopo una blittata c'è subito una routine che impiega il blitter, il processore dovrà comunque attendere che il blitter termini il suo compito, e non avremo dunque nessun vantaggio. Quindi un criterio da seguire per ottimizzare i programmi è quello di mettere, quando possibile, le routine che usano il blitter "distanti", ovvero intervallate da altre routine che non ne facciano uso, in modo che il blitter e il processore procedano in parallelo. C'è da dire però che questo criterio è valido soprattutto su macchine dotate di memoria fast, in quanto se il processore deve accedere alla memoria chip si generano dei conflitti nell'accesso alla memoria, di cui parleremo meglio tra un attimo.

Per il momento notiamo un'altra cosa sugli esempi `lezione10e2.s` e `lezione10e2r.s`: il blitter impiega circa lo stesso tempo per la cancellazione (schermo verde) e per il disegno (schermo rosso). Se ci pensate bene, questo fatto dovrebbe sembrarvi strano: infatti è vero che le 2 blittate hanno la stessa dimensione ma bisogna considerare che la cancellazione è una blittata che usa un solo canale, mentre la copia ne usa 2. È chiaro che all'aumentare del numero di canali, aumenta il numero di words lette e scritte dal blitter, quindi la blittata dovrebbe richiedere più tempo. Guardate però l'esempio `lezione10e3.s`.

Questo esempio è simile ai precedenti, ma invece di fare una semplice copia della figura effettua un operazione di OR tra la figura e un plane azzerato. Naturalmente l'effetto è sempre lo stesso, ma potete notare come ora la routine, che effettua una blittata a 3 canali (D=A OR B) sia notevolmente più lenta. La velocità dipende da quali e quanti canali sono usati in una maniera abbastanza complicata, che può essere sintetizzata dalla seguente tabella:

```
 bit 8-11
    di        Canali
 BLTCON0      usati         Sequenza di accesso alla memoria
---------    --------    ---------------------------------------
    F        A B C D     A0 B0 C0 -  A1 B1 C1 D0 A2 B2 C2 D1 D2
    E        A B C       A0 B0 C0 A1 B1 C1 A2 B2 C2
    D        A B   D     A0 B0 -  A1 B1 D0 A2 B2 D1 -  D2
    C        A B         A0 B0 -  A1 B1 -  A2 B2
    B        A   C D     A0 C0 -  A1 C1 D0 A2 C2 D1 -  D2
    A        A   C       A0 C0 A1 C1 A2 C2
    9        A     D     A0 -  A1 D0 A2 D1 -  D2
```

```
8       A           A0 -  A1 -  A2
7         B C D     B0 C0 -  -  B1 C1 D0 -  B2 C2 D1 -  D2
6         B C       B0 C0 -  B1 C1 -  B2 C2
5         B   D     B0 -  -  B1 D0 -  B2 D1 -  D2
4         B         B0 -  -  B1 -  -  B2
3           C D     C0 -  -  C1 D0 -  C2 D1 -  D2
2           C       C0 -  C1 -  C2
1             D     D0 -  D1 -  D2
0       nessuno     -  -  -  -
```

Questa tabella mostra per ogni combinazione di canali attivi, la sequenza di accessi alla memoria operata dal blitter, nel caso di una blittata di 3 words. Per ogni accesso è indicato il canale che lo effettua, e i trattini indicano cicli di bus non sfruttati dal blitter. Per esempio la stringa:

```
A0 B0 -  A1 B1 -  A2 B2
```

Indica che prima accede al bus il canale A (A0) poi il B (B0), poi il blitter non utilizza un ciclo di bus (permettendo al processore di accedere alla memoria), poi tocca di nuovo al canale A (A1) e così via.

La tabella riportata in realtà è solo indicativa, perché non tiene conto di molti fattori, quali l'utilizzo di modi speciali del blitter e la competizione con il processore e con gli altri canali DMA (consultate al riguardo la lezione 8). Ciononostante è molto utile per avere un idea di quali sono le combinazioni di canali migliori. Tenete presente che questa tabella è relativa ad una blittata di 3 words. Per blittate di più words il blitter ripete tante volte la sequenza di accessi che nella tabella sono "al centro". Per esempio, una blittata di 5 words che utilizza i canali A e D ha la seguente sequenza:

```
A0 -  A1 D0 A2 D1 A3 D2 A4 D3 A5 D4 -  D5
```

Lo studio della tabella ci consente alcune osservazioni interessanti. Se guardiamo la sequenza relativa all'uso del solo canale D, vediamo che il blitter sfrutta il bus un ciclo si e uno no. Al contrario, quando vengono usati i canali A e D, il blitter sfrutta (tranne che nei casi della prima e dell'ultima word) tutti i cicli di bus. Questo fatto ci spiega come mai negli esempi la routine di cancellazione (canale D) ha circa la stessa velocità della routine di disegno (canali A e D). Notate però che se facciamo una copia da B a D, le cose vanno diversamente.

Lo potete vedere in pratica nella `lezione10e4.s`.

Analogamente, consultando la tabella, si vede che nel caso di blittate con 2 sorgenti conviene usare A e B oppure A e C, ma non B e C perché vengono sprecati più cicli.

Dovete comunque ricordare che la velocità del blitter dipende anche dagli eventuali conflitti con altri canali DMA (video, audio, copper, processore) che possono "rubargli" cicli ritardandolo. Infatti, come abbiamo spiegato nella lezione 8, il blitter nell'accesso al bus ha priorità solo sulla CPU. Questo significa che se un altro dispositivo (es. il copper) vuole accedere alla RAM contemporaneamente al blitter, la precedenza spetta all'altro dispositivo. L'unico fesso che da la precedenza al blitter è il processore. Anche qui però la priorità non è totale. Infatti il blitter, dando prova di grande generosità, se si accorge che il processore per 3 volte consecutive ha provato ad accedere al bus ma non c'è riuscito perché qualcun altro gli ha preso la precedenza, gli dice: "Passa tu per stavolta, vah" e gli concede il bus per un ciclo.

Questo meccanismo riduce la possibilità che in casi di sovraccarico del DMA il processore sia bloccato in attesa del bus troppo a lungo. È comunque possibile reprimere i moti di generosità del blitter. Settando a 1 il bit 10 (detto `blitter_nasty`, cioè blitter cattivo) del registro `DMACON` il blitter non si comporterà più in questo modo, ma si prenderà la precedenza sul processore

ogni volta. Nel caso in cui le routine del nostro programma utilizzino tutte il blitter, quindi il processore non fa altro che caricare i registri e mettersi in attesa, conviene senz'altro settare a 1 tale bit. Ovviamente questo discorso ha senso in caso il cui il programma sia contenuto in memoria chip ed in assenza di caches, perché in caso contrario non si verificano conflitti tra il processore e il blitter per l'accesso alla RAM. Un esempio sul bit Blitter Nasty si trova in `lezione10e5.s`.

Per ottimizzare al massimo l'uso del blitter, dovete velocizzare al massimo la scrittura dei registri ad esso relativi. Negli esempi che abbiamo fatto sinora e anche in quelli che faremo nel resto della lezione infatti, per aumentare la chiarezza, non abbiamo ottimizzato la scrittura dei registri come avremmo potuto. Durante una blittata, gli unici registri che variano sono i registri `BLTxPT` e il `BLTSIZE`. I registri `BLTCONx`, `BLTxMOD` e `BLTxWM` rimangono costanti. Ciò significa che se il contenuto di tali registri non viene modificato da altre routine, non serve riscriverli all'inizio di ogni blittata. Un accorgimento da adottare per ottimizzare le routine nel caso in cui ci sono loop di blittate è quello di porre i valori da scrivere nei registri blitter in registri del processore, e sostituire all'interno del loop le `MOVE.W #YYY,$DFFxxx` con delle `MOVE.W Dx,$DFFxxx` che sono più veloci. Queste ottimizzazioni nella scrittura dei registri prese una per una danno incrementi di velocità davvero minimi, che con il copper monitor è difficile notare. Però in una demo con tanti effetti complessi, messe insieme hanno il loro peso.

A titolo di esempio guardate il listato `lezione10e6.s` che è una versione ottimizzata con questi trucchi di `lezione10c3.s`.

10.4 Il double buffering

Tutti gli esempi che abbiamo visto finora relativi ai bobs avevano sempre un solo bob che si muoveva sullo schermo. Proviamo ora a metterne di più. Per esempio, proviamo ad applicare la tecnica dello sfondo "finto": usiamo un bitplane per lo sfondo e 3 planes dove muovere i bobs. Poiché tutti i bobs si muovono sugli stessi bitplanes dovremo comunque disegnarli usando la tecnica del bitplane maschera. Avremo comunque il vantaggio di non dover salvare e ripristinare lo sfondo, perch i bitplanes dei bob inizialmente sono azzerati. Sarà quindi sufficiente cancellare questi plane ad ogni frame, prima di ridisegnare i bobs nelle nuove posizioni. Questa tecnica è applicata nell'esempio `lezione10f1.s`.

Eseguendo questo programma avrete però una brutta sorpresa: i bobs vengono disegnati correttamente solo nella parte bassa dello schermo, mentre in alto non vengono disegnati correttamente. Come mai? C'è qualche bug nelle nostre routines? No, le nostre routines vanno bene. Il problema è che sono troppo lente. Come ben sapete, mentre il nostro programma viene eseguito, il pennello elettronico disegna l'immagine sullo schermo.

Per far apparire un immagine stabile, si cerca di modificare lo schermo (cioè cancellare, disegnare bobs, linee ecc.) durante il Vertical Blank, ossia nel periodo di tempo in cui il pennello elettronico è inattivo. Se però dobbiamo fare molte modifiche sullo schermo, può accadere che le nostre routines non siano abbastanza veloci da svolgere il proprio compito durante il Vertical Blank. È appunto quello che succede in questo caso. Aumentando il numero di bobs, aumenta il tempo necessario a disegnarli e di conseguenza non si riesce più a farlo durante il Vertical Blank. Il risultato è che a volte i bobs vengono disegnati sullo schermo DOPO che il pennello elettronico ha disegnato quella parte di schermo, e quindi i bobs non vengono visualizzati. Poiché il pennello elettronico va dall'alto verso il basso, più i bob sono disegnati in alto e più spesso ciò accade. Se guardate attentamente l'esempio, vedrete che la zona di schermo nella quale tutti i bobs vengono disegnati bene, è quella che viene visualizzata DOPO che le routines di disegno hanno finito il loro lavoro, come viene evidenziato dal copper monitor.

La tecnica del "double buffering" ci consente di risolvere questo problema. Si tratta di una tecnica di uso generale che potete impiegare con qualsiasi effetto, non solo con i bobs. In particolare lo useremo per le routines 3d. Questa tecnica consiste nell'utilizzare due schermi (detti appunto buffer) invece che uno solo. I due buffer vengono visualizzati alternativamente, un fotogramma l'uno e un fotogramma l'altro. Mentre viene visualizzato uno dei buffer, noi possiamo disegnare liberamente sull'altro, senza preoccuparci della stabilità, visto che l'immagine che viene visualizzata è quella del primo buffer che noi non modifichiamo. Quando si verifica il successivo Vertical Blank, i 2 buffer vengono scambiati. Quello sul quale noi abbiamo disegnato in precedenza viene visualizzato, mostrando le modifiche che abbiamo fatto, mentre il buffer che prima era stato visualizzato è ora a nostra disposizione per disegnarci sopra. Ripetendo lo scambio ad ogni Vertical Blank, avremo sempre a disposizione un buffer non visualizzato sul quale disegnare, senza preoccuparci di quello che fa il pennello elettronico. Grazie a questa tecnica, l'unica limitazione di tempo delle nostre routines di disegno è che esse devono concludersi prima che il pennello elettronico raggiunga la fine dello schermo. Questo ci da un tempo pari ad 1/50-esimo di secondo (in Pal, 1/60 in NTSC).

10.5 Uso dei canali Blitter non attivati

Vi sono casi in cui è utile far "partecipare" alla blittata anche i canali non attivi. Per capire bene cosa significa dovete sapere ancora una cosa sul blitter. Quando un canale di ingresso (A, B o C) è attivo, legge words dalla memoria. Ogni word dopo essere stata letta viene copiata in un apposito registro, detto registro dati del blitter. Ogni canale ha il suo registro dati, nel cui nome compare la lettera che identifica il canale: abbiamo dunque `BLTADAT` (canale A, *$DFF074*), `BLTBDAT` (canale B, *$DFF072*), `BLTCDAT` (canale C, `$DFF070`) e `BLTDDAT` (canale D `$DFF000`). La word dal registro dati viene successivamente sottoposta ad operazioni logiche con le words provenienti dagli altri canali, e il risultato viene scritto in memoria attraverso il canale D. Facciamo un esempio per capire bene. Consideriamo il caso di una blittata che esegua un AND tra i canali B e C. All'interno del blitter accadono le seguenti cose:

1. Il canale B legge una word e la copia in `BLTBDAT`
2. Il canale C legge una word e la copia in `BLTCDAT`
3. Viene eseguito un AND tra il contenuto di `BLTBDAT` e quello di `BLTCDAT`
4. Il risultato viene scritto attraverso il canale D
5. Si ripetono i passi da 1 a 4 per le successive words.

In realtà le cose funzionano un pò diversamente, perché alcune operazioni sono eseguite in parallelo per velocizzare il blitter, ma a livello logico le cose funzionano così, ed è quello che ci serve sapere. Che succede quando un canale è disabilitato? Naturalmente esso non leggerà nulla dalla memoria, pertanto il registro `BLTxDAT` corrispondente non verrà cambiato. Il contenuto di tale registro viene conservato, e può comunque essere usato in operazioni logiche. Inoltre tale registro può essere scritto anche dalla CPU, il che ci permette di settarlo a valori opportuni (non il registro `BLTDDAT`!). La situazione è simile a quella che abbiamo visto nella lezione 7 per gli sprite. Anche gli sprite hanno dei canali DMA (i registri `SPRxPT`) che copiano i dati letti in registri dati (`SPRxDAT`). In alcune applicazioni però è utile scrivere nei registri dati direttamente con il processore (o con il copper). Vediamo ora l'utilità di questa caratteristica del blitter. Per esempio consideriamo il caso in cui vogliamo riempire una serie di locazioni di memoria con un valore

costante, per esempio per disegnare sullo schermo un rettangolo non pieno, ma "a righine", o come dicono i grafici con un "pattern" (cioè una trama).

Potremo risolvere il problema memorizzando il nostro rettangolo nella sezione dati del nostro programma e copiandolo con il blitter, esattamente come se fosse una figura come le altre. Una soluzione migliore, però, ci viene offerta dalla possibilità di disabilitare i canali del blitter. Per risolvere il problema possiamo infatti eseguire una copia dal canale A al D, TENENDO IL CANALE A disabilitato, e scrivendo il "pattern" nel registro `BLTADAT`. In questo modo otteniamo 2 vantaggi: non dobbiamo memorizzare il rettangolo tra i dati del nostro programma, quindi risparmiamo memoria, e, poiché il canale A è disabilitato, effettuiamo meno accessi alla memoria di quanti ne faremmo in caso di copia normale da A a D, dando pertanto al processore più possibilità di accesso alla RAM.

Per vedere in pratica questa applicazione caricate la `lezione10g1.s`.

È possibile applicare questa tecnica non solo per semplici copie di un valore costante, ma anche in operazioni logiche più complesse nelle quali un operando sia costante.

Trovate 2 esempi in `lezione10g2.s` e `lezione10g3.s`.

10.6 Il flag Zero e le collisioni

Questa è l'ultima caratteristica hardware del blitter da spiegare! Il blitter ha un flag, detto flag Zero, che ha un funzionamento analogo al flag Zero del processore. Questo flag è il bit 13 del registro `DMACONR`. Se una blittata produce come risultato TUTTI ZERI, il flag Zero viene settato a UNO. Al contrario, se almeno un bit in una delle word risultato ha valore 1, il flag assume valore ZERO. Il flag si comporta in questo modo anche nel caso in cui il risultato della blittata NON viene scritto in memoria, cioè quando il canale D è disabilitato.

Questo fatto è molto utile perché ci aiuta a rilevare collisioni tra un bob e un disegno sullo schermo (che può essere un altro bob già disegnato). Supponiamo per il momento di lavorare con immagini ad un solo bitplane. Per rilevare le collisioni effettuiamo (con il blitter) un'operazione di AND tra il bob e la parte di schermo su cui il bob si dovrebbe posizionare, MA non scriviamo il risultato da nessuna parte. Questa blittata serve solo per testare la collisione. Cosa succede quando eseguiamo un AND? Come sapete il risultato di un AND tra 2 bit è 1 solo nel caso in cui entrambi i bit operandi valgono 1. Nel nostro caso significa che un bit del risultato può valere 1 SOLO nel caso in cui coincidono nella stessa posizione un bit del bob di valore 1 e un bit dell'immagine di valore 1. Ma ciò vuol dire che tali bit producono una collisione. Quindi se c'è una collisione, almeno un bit del risultato avrà valore UNO, e in corrispondenza il flag Zero assumerà valore ZERO. Al contrario, se non si verifica collisione, nessun bit del bob coincide con un bit dello sfondo, quindi l'AND vale SEMPRE ZERO, e quindi il flag Zero assume valore UNO. Quindi il flag Zero ci può segnalare quando c'è una collisione e quando no.

Quando abbiamo a che fare con immagini con più bitplanes, le cose si complicano in quanto potrebbe accadere che si verifica una collisione tra 2 pixel di colori diversi che considerati plane per plane non coincidono. Per esempio, se si verifica una collisione tra un pixel di colore 1 (plane 1 = 1 e tutti gli altri a 0) e un pixel di colore 2 (plane 2 = 1 e tutti gli altri a 0) facendo un AND plane a plane, il risultato è sempre 0. In questi casi conviene usare i bitplane maschera. Essi infatti hanno un bit a 1 ogni volta che il corrispondente pixel del bob ha un colore diverso dallo sfondo. Quindi facendo l'AND tra 2 bitplane maschera si rilevano collisioni qualsiasi sia il colore dei pixel (è come rilevare la collisioni tra le "ombre" dei 2 bob, che sono immagini ad 1 plane).

Potete vedere un esempio in `lezione10h1.s`.

10.7 Il SinusScroll

Quasi sicuramente ognuno di voi sa cosa è un sine-scroller. Si tratta di uno scrolltext che quando scorre sullo schermo sale e scende, in modo da formare una sinusoide. Prima di iniziare a spiegare come funziona il sine-scroller, è bene puntualizzare alcune cose. Innanzitutto, il tempo. Un sine-scroller è una routine molto lenta. Un buon sine-scroller può prendere anche più di un quarto del tempo disponibile in un fotogramma. Per sistemi privi di caches e di memoria fast (in pratica l'Amiga 500 e 600) è estremamente utile settare a 1 il flag `BLITTER_NASTY`, che da al blitter la priorità assoluta sul 68000 per migliorare le prestazioni della routine. Inoltre si deve anche considerare "la qualità" del sine-scroller che si vuole ottenere. Con questo si intende dire quanti pixel dovrebbero essere mostrati in ogni posizione sinusoidale. Un sine-scroller da 1 pixel è quello che ha l'aspetto più fluido, ma anche quello che richiede più tempo.

Non aspettatevi di avere tempo per altri effetti se usate uno schermo non "double bufferizzato". D'altro canto già un sine-scroller da 4 pixel inizia a sembrare molto "pixelloso". Per questo spiegheremo inizialmente come realizzare un sine-scroller da 2 pixel, e poi le variazioni da apportare per le versioni da 1 e da 4 pixel. Siete un pò confusi? Vediamo con un esempio che significa esattamente ciò che abbiamo detto a proposito della qualità.

Immaginate che la figura riportata sotto sia la lettera A di un font bitmap:

```
.**************.
****************
****************
******....******
*****......*****
****************
****************
****************
*****......*****
*****......*****
*****......*****
*****......*****
*****......*****
*****......*****
*****......*****
................
```

Fig. 31 lettera A

Un "*" indica un bit settato a 1, un "." è un bit azzerato. Il carattere "A", quando viene scrollato normalmente in orizzontale appare sempre uguale a come è memorizzato nei dati del font. In un sine scroller noi non vogliamo questo. Noi vogliamo cambiare che le colonne di pixel che compongono il carattere, in modo che assumano differenti posizioni verticali, in base ai valori di una sinusoide. In un sine-scroller da 1 pixel, ogni colonna di pixel assume una differente posizione verticale. Invece, in un sine-scroller da 2 pixel, le colonne di pixel sono accoppiate a 2 a 2, e ogni coppia di colonne assume una posizione verticale diversa dalle altre coppie. Un sine-scroller da 1 pixel deforma il carattere A nel modo illustrato dalla figura seguente.

```
.
 **
 ***
 ****
 *****
 ******
 *******
 ********
```

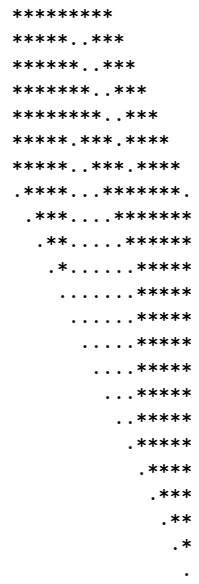


Fig. 31 lettera A deformata da un sine-scroller da 1 pixel

Come vedete ogni colonna di pixel si trova in una posizione verticale diversa dalle altre. Un sine-scroller da 2 pixel invece produce il seguente risultato:

Fig. 32 lettera A deformata da un sine-scroller da 2 pixel

Come vedete coppie di colonne adiacenti hanno la stessa posizione verticale. In un sine-scroller da 4 pixel, come avrete capito, le colonne di pixel son raggruppate a 4 a 4 e ogni gruppo assume una posizione diversa da un altro gruppo. Ora dovreste aver capito cosa si intende per sine-scroll da “1 pixel” o da “2 pixel”.

Il metodo per realizzare un sine-scroller è molto semplice. Si parte da una normale routine di text scrolling, come quelle che abbiamo visto in precedenza. Però, invece di disegnare e scrollare il nostro testo sullo schermo visibile, lo facciamo in un buffer di dati allocato da qualche parte in memoria. Questo buffer di scroll non è mai visibile. Da questo buffer noi prendiamo delle “fettine” verticali di scroller e le copiamo nello schermo visibile. Ogni “fettina” viene copiata ad una differente posizione verticale, in base ad i valori della sinusoide. Lo spessore delle “fettine” determina la qualità del sine-scroller. Se esse sono spesse 1 pixel, abbiamo un sine scroller da 1 pixel, se sono spesse 2 pixel abbiamo una routine da 2 pixel e così via. Vediamo più in dettaglio come effettuare la copia delle “fettine”.

Poiché le fettine sono molto sottili, faremo blittate larghe una sola word. Per selezionare all’interno della word solo la fettina (cioè solo le colonne di pixel) che ci interessano, useremo uno dei registri maschera del canale A (questo significa che siamo obbligati ad usare il canale A per la lettura) che ci permette di cancellare tutte le colonne di pixel che non fanno parte della fettina che ci interessa. Naturalmente, il valore della maschera varierà a seconda della “fettina” da leggere. La scrittura, come abbiamo già detto, avviene ogni volta ad una diversa posizione verticale. Quando effettuiamo la scrittura, non basta fare una semplice copia da A a D: se facessimo in questo modo, copiando una “fettina” cancelleremmo una parte delle “fettine” copiate in precedenza che appartengono alla stessa word della “fettina” attuale. Infatti, anche se le altre "fettine" non si sovrappongono alla nostra (perché si trovano una accanto all’altra) siccome la nostra blittata è larga una word, con una copia semplice copieremmo sullo schermo anche le colonne di pixel azzerate dalla maschera che si trovano a fianco della “fettina” attuale. Per risolvere questo problema, facciamo un OR tra la nostra word e lo sfondo sulla quale la scriviamo. In questo modo i pixel azzerati della word attuale non sovrascrivono quelli dello sfondo.

Per realizzare il sine-scroller è sufficiente copiare dal buffer allo schermo, mediante questo procedimento, tutto lo scrolltext una “fettina” alla volta. Ovviamente tutta la procedura deve essere ripetuta ad ogni fotogramma, perché lo scrolltext si è spostato e ogni volta, prima di effettuarla, è necessario cancellare lo schermo. Notate che maggiore è l’ampiezza della sinusoide e maggiore è l’area di schermo coinvolta nell’operazione, e che dobbiamo ogni volta cancellare. Quindi conviene usare una sinusoide poco ampia per migliorare le prestazioni.

In `lezione10i1.s` e `lezione10i2.s` troverete rispettivamente un sine-scroller da 2 pixel e uno da 1 pixel.

10.8 Animazione

Concludiamo la lezione con una breve spiegazione su come creare animazioni con il blitter. Un animazione è costituita da una serie di immagini (fotogrammi) che devono essere mostrati secondo una certa sequenza. Di solito tra un fotogramma e l’altro non varia tutta l’immagine, ma solo delle parti di essa. Per esempio potremmo avere un castello con delle bandiere che si muovono a causa del vento. Chiaramente solo la parte di schermo sulla quale sono disegnate le bandiere cambia tra un fotogramma e l’altro.

Per risparmiare memoria non conviene memorizzare tutte le immagini dell’animazione: basta memorizzare la prima immagine e poi i “pezzi” delle altre immagini che contengono le differenze con la prima. In questo modo per realizzare l’animazione basta copiare i nuovi “pezzi” di immagine sulla vecchia. Per questo scopo ci è molto utile il blitter che come sapete è molto più veloce

del 68000 (di base) nel copiare dati. In sostanza per realizzare un animazione bisogna fare delle copie con il blitter, cosa di cui ormai siamo maestri. Le animazioni possono essere divise in due tipi a seconda di come è strutturata la sequenza di fotogrammi. Nelle animazioni del primo tipo, dette animazioni *cicliche*, i fotogrammi vengono disegnati uno dopo l'altro in base ad un ordine predeterminato. Dopo che è stato disegnato l'ultimo, l'animazione continua ripartendo dal primo fotogramma. Anche nelle animazioni del secondo tipo (animazioni *avanti-indietro*) i fotogrammi vengono disegnati in base ad un ordine. Però, dopo che è stato disegnato l'ultimo fotogramma, l'animazione continua ridisegnando i fotogrammi in ordine inverso, dal penultimo fino a tornare al primo. A questo punto l'animazione procede di nuovo in ordine diretto fino all'ultimo, poi ancora in ordine inverso e così via. In base al tipo di animazione si dovrà usare una diversa routine di gestione dei fotogrammi.

Vi presentiamo 2 esempi di animazione (uno per ogni tipo) nei listati `lezione10l1.s` e `lezione10l2.s`.

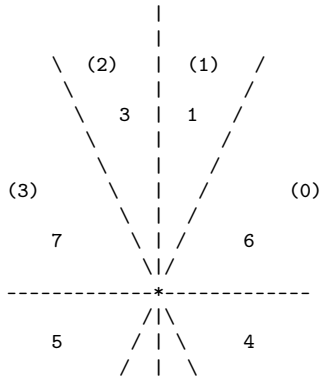
È possibile anche realizzare dei bob animati. Si tratta di bob che cambiano forma ogni volta che vengono disegnati. Naturalmente anche per i bob abbiamo a disposizione una serie di fotogrammi che vengono presentati in sequenza, in base ad una delle 2 tecniche di cui abbiamo parlato prima. Ogni volta che il bob deve essere disegnato bisogna utilizzare un figura diversa. È molto comodo dunque poter disporre di una routine universale, capace di disegnare come un bob qualsiasi figura, di dimensioni variabili.

Troverete una routine del genere per schermi in formato normale nell'esempio `lezione10m1.s` e per schermi in formato INTERLEAVED nell'esempio `lezione10m2.s`.

10.9 I modi speciali del Blitter

In aggiunta a tutte le funzioni descritte fin qui, il blitter ha anche la possibilità di disegnare linee e quella di "riempire" delle aree, cioè di settare a 1 tutti i bit di una determinata regione di un bitplane. Queste capacità aggiuntive sono ottenute mediante degli speciali modi di funzionamento del blitter.

Iniziamo a parlare del tracciamento di linee. Quando il blitter opera in modo di tracciamento di linee (detto *line-mode*) esso disegna una linea da un punto dello schermo (che chiamiamo P1) a un altro (che chiamiamo P2). Indichiamo con X1 e Y1, rispettivamente l'ascissa e l'ordinata di P1, e con X2 e Y2 l'ascissa e l'ordinata di P2. In "line-mode" molti registri funzionano in maniera completamente differente rispetto a quanto visto finora ed è necessario settarli in maniera opportuna. Alcuni settaggi dipendono dalla posizione di P1 e P2. Prima di descrivere l'uso dei registri è necessario fare delle considerazioni preliminari. Durante il tracciamento il blitter considera lo schermo diviso in *ottanti* rispetto al punto P1. Per capire meglio guardate la seguente figura:



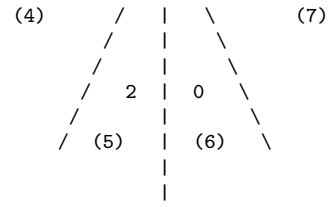


Fig. 1 Ottanti

Nella figura l'asterisco (*) rappresenta il punto P1. Il blitter considera lo schermo diviso nelle 8 regioni (dette ottanti) rappresentate in figura. La linea da tracciare appartene ad uno degli ottanti, quello nel quale si trova P2. I numeri tra parentesi servono per numerare gli ottanti secondo la notazione di solito usata da noi "umani" (cioè in senso antiorario). Il blitter invece li numera in una maniera un pò strana che è indicata dai numeri senza parentesi. Di questa divisione dello schermo terremo conto in seguito. Dobbiamo inoltre definire alcune quantità che dovremo usare per preparare la blittata. Chiamiamo DiffX la differenza tra le ascisse di P2 e P1, cambiata di segno nel caso in cui venga negativa, in modo che sia comunque positiva. In formule poniamo:

```
DiffX = abs(X2 - X1)
```

dove "abs" indica la funzione che calcola il valore assoluto di un numero. Facciamo la stessa cosa con le ordinate ponendo:

```
DiffY = abs(Y2 - Y1).
```

A questo punto definiamo DX e DY rispettivamente come massimo e minimo tra DiffX e DiffY. In formule:

```
DX = max(diffX,diffY)
DY = min(diffX,diffY).
```

Cominciamo ora a vedere come vanno settati i registri del blitter, a cominciare da `BLTCON1` che permette di attivare il line-mode. Il bit 0 di `BLTCON1` serve appunto a questo scopo. Quando è settato a 1 si attiva il line-mode. Il bit 1 permette di disegnare delle linee "speciali" che permettono il successivo riempimento di aree del blitter. Ne parleremo più avanti, per ora lo lasciamo a 0 (linee normali). Nei bit 2,3 e 4 va scritto il numero dell'ottante nel quale si trova il punto P2. Naturalmente dovremo usare la numerazione del blitter. Per convertire facilmente la normale numerazione in senso antiorario in quella usata dal blitter potete consultare la seguente tabella:

Valore Bit di BLTCON1	Numero Ottante
4 3 2	
- - -	
1 1 0	0
0 0 1	1
0 1 1	2
1 1 1	3
1 0 1	4
0 1 0	5
0 0 0	6
1 0 0	7

Il bit 6 di BLTCON1 (detto bit SIGN) va settato a 1 nel caso in cui risulti che 4*DY-2*DX<0. Altrimenti (cioè se 4*DY-2*DX>0) va settato a 0. I bit da 12 a 15 di BLTCON1 contengono la posizione di partenza del "pattern" della linea. Infatti è possibile disegnare non solo linee "solide", ma anche linee tratteggiate, mediante un "pattern" che viene ripetuto lungo tutta la linea (abbiamo già visto esempi di pattern nella lezione 9). I bit da 12 a 15 di BLTCON1 indicano il pixel a partire dal quale deve essere usato il pattern. Naturalmente (abbiamo solo 4 bit) deve essere uno dei primi 16 pixel della linea. Tutti gli altri bit di BLTCON1 vanno lasciati a 0.

Veniamo ora a BLTCON0. Il byte basso di tale registro (LF, quello dei minterms) permette di selezionare 2 diverse modalità di disegno. Ponendo LF=$4A viene eseguita un operazione di OR-esclusivo tra la linea e lo sfondo su cui viene tracciata. In pratica i pixel attraversati dalla linea vengono invertiti. Ponendo invece LF=$CA viene eseguita un operazione di OR semplice tra la linea e lo sfondo. In pratica i pixel attraversati dalla linea vengono accesi. I canali da attivare per la blittata sono A, C e D. Quindi i bit 8, 9 e 11 devono essere settati a 1, mentre il 10 a 0. I bit da 12 a 15 di BLTCON0 devono invece contenere i 4 bit meno significativi (cioè più bassi) di X1, l'ascissa del punto P1. I settaggi degli altri registri sono fortunatamente più semplici. I registri BLTAFWM e BLTALWM devono essere settati al valore $FFFF (non mascherano nulla). Il registro BLTADAT deve contenere invece il valore $8000, che rappresenta il pixel da disegnare. Il registro BLTBDAT invece contiene il "pattern" della linea, a cui abbiamo accennato prima. Un valore $FFFF fa disegnare una linea continua.

Nel tracciamento di linee viene usato solo la parte bassa di BLTAPT, ovvero solo il registro 16 bit BLTAPTL, che deve essere settato al valore 4*DY-2*DX. Il registro BLTAMOD, invece va settato al valore 4*DY-4*DX. Il registro BLTBMOD va settato al valore 4*DY. I registri BLTCPT e BLTDPT devono contenere l'indirizzo della word dello schermo che contiene il pixel P1. I registri BLTCMOD e BLTDMOD devono contenere la larghezza dello schermo espressa in bytes.

Infine il registro BLTSIZE deve essere settato in maniera da eseguire una blittata larga 2 words e alta un numero di linee pari a DX+1. Il che vuol dire che i bit da 0 a 5 devono contenere il numero 2 mentre i bit da 6 a 15 il valore DX+1. Come accade di solito, scrivendo nel registro BLTSIZE si attiva il blitter. Per questo motivo, tale registro deve essere scritto per ultimo.

In sintesi, i valori da caricare nei registri sono:

```
BLTADAT = $8000
BLTBDAT = pattern linea ($FFFF per una linea solida)

BLTAFWM = $FFFF
BLTALWM = $FFFF

BLTAMOD = 4 * (dy - dx)
BLTBMOD = 4 * dy
BLTCMOD = larghezza del bitplane in byte
BLTDMOD = larghezza del bitplane in byte

BLTAPT = (4 * dy) - (2 * dx)
BLTBPT = non usato
BLTCPT = puntatore alla word che contiene il primo pixel della linea
BLTDPT = puntatore alla word che contiene il primo pixel della linea

BLTCON0 bit 15-12 = i 4 bit più bassi di X1
BLTCON0 bit 11 (SRCA), 9 (SRCC), e 8 (SRCD) = 1
BLTCON0 bit 10 (SRCB) = 0
BLTCON0 LF byte di controllo = $4A (per linea in EOR)
                             = $CA (per linea in OR)

BLTCON1 bit 0 = 1
BLTCON1 bit 4-2 = numero ottante (dalla tavola)
BLTCON1 bit 15-12 = bit iniziale per pattern linea
```

```
BLTCON1 bit 6 = 1 se (4 * dy) - (2 * dx)) < 0
              = 0 altrimenti
BLTCON1 bit 1 = 0 (per linee normali)
              = 1 (per linee speciali per il fill)

BLTSIZE bit 15-6 = dx + 1
BLTSIZE bit 5-0 = 2
```

Un esempio di tracciamento di linea è contenuto in `lezione10n.s`. Si tratta di una routine semplificata al massimo, senza ottimizzazioni particolari, per facilitare la comprensione a scapito della velocità di esecuzione.

10.10 Modo di Riempimento aree

Oltre a copiare dati, il blitter può simultaneamente eseguire un'operazione di fill (riempimento) durante la copia. Questo modo può essere attivato con una qualsiasi blittata standard (COPA, AND, OR, ecc.) e viene effettuato **dopo** tutte le altre operazioni che già conoscete (shift, mascheramenti, ecc.). Per capire come funziona il riempimento immaginate che il blitter scriva in uscita un bit alla volta (il che è falso, come sapete, perché scrive sempre UNA WORD alla volta) e che stia effettuando una semplice operazione di copia. Finché legge bit di valore 0, li copia normalmente. Ad un certo punto gli arriva un bit di valore 1. Lo copia ugualmente nell'uscita, ma a partire da questo momento, invece di continuare a copiare i bit seguenti, manda in uscita tutti bit di valore 1. Quando però legge un secondo bit di valore 1, riprende il comportamento normale. Quando poi legge un terzo bit di valore 1, ricomincia a mandare degli 1 in uscita, fino al successivo 1 in ingresso, e così via. Vediamo cosa succede ai dati copiati, mostrando una sequenza di bit in ingresso di esempio e la corrispondente uscita:

```
ingresso        00010001001001000100000100011001 0010
uscita          00011111001111000111111100011001 1110
```

In pratica i bit di valore 1 sono considerati i bordi dell'area e quindi il blitter riempie (cioè setta ad 1) i bit compresi dentro i bordi. Vediamo ora i dettagli tecnici del fill-mode. Come abbiamo già detto esso può essere usato in combinazione con una qualsiasi blittata, in quanto il riempimento viene effettuato dopo che i dati letti dalle 3 sorgenti sono stati combinati tra loro in base alla funzione logica selezionata dai minterms. Il fill-mode, però può essere usato solo con blittate effettuate in modo discendente. Vi sono 2 diversi tipi di fill, detti inclusivo ed esclusivo. Ogni tipo di fill ha un suo bit di abilitazione. Per attivare il fill-mode bisogna settare ad 1 uno dei 2 bit di abilitazione. Non è possibile attivare contemporaneamente i 2 diversi fill. Vediamo le differenze tra i 2 tipi di fill. Il modo di riempimento inclusivo riempe tra le linee, lasciandole intatte. Il modo esclusivo riempe tra le linee, ma pur lasciando la linea di delimitazione di destra, cancella quella di sinistra. Dunque il fill esclusivo produce forme riempite un pixel più strette dello stesso pattern (contorno) riempito con fill inclusivo.

Per esempio, il pattern:

```
00100100-00011000
```

riempito con fill inclusivo, produce:

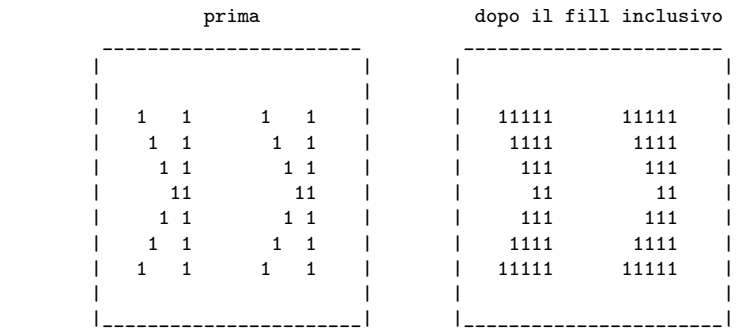
```
00111100-00011000
```

con fill esclusivo, il risultato sarebbe:

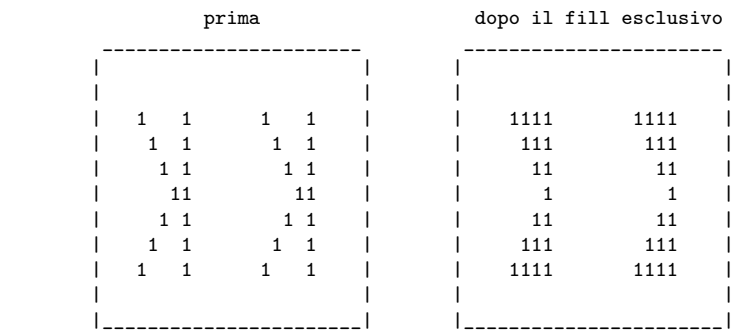
```
00011100-00001000
```

(Naturalmente, i riempimenti sono sempre fatti su piene word di 16-bit.) Facciamo un altro esempio con l'aiuto di disegni:

`fill inclusivo:`



`fill esclusivo:`



come potete vedere, con il fill esclusivo sono state cancellate le linee sinistre della figura. In questo modo si ottengono figure con bordi più affilati. Il bit di abilitazione del fill inclusivo è il bit 3 di `BLTCON1`, mentre quello del fill esclusivo è il bit 4, sempre di `BLTCON1`.

C'è un altro bit che serve a controllare il riempimento. Si tratta del bit 2 di `BLTCON1` (detto `FILL_CARRYIN`) che, quando viene settato ad 1, forza il riempimento delle zone esterne alle linee, anziché di quelle interne. Torniamo al primo esempio che abbiamo fatto e vediamo cosa succede alla nostra riga di bit quando il bit `FILL_CARRYIN` è settato a 1. La riga di partenza era:

```
00100100-00011000
```

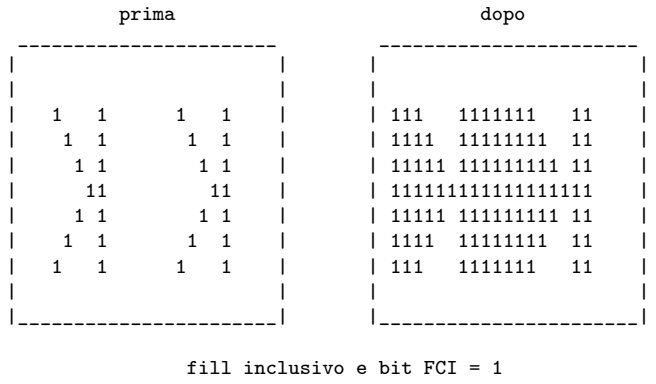
Con fill inclusivo e `FILL_CARRYIN=1`, l'output sarebbe:

```
11100111-11111111
```

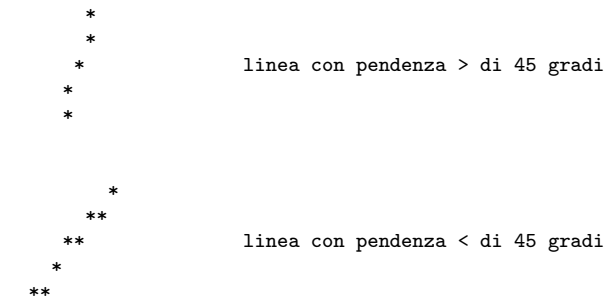
Con fill esclusivo e `FILL_CARRYIN=1`, l'output sarebbe:

```
11100011-11110111
```

Vediamo cosa succede nel caso del secondo esempio con fill inclusivo e `FILL_CARRYIN=1`.



Il fill-mode viene usato sopratutto per il riempimento di poligoni. I bordi dei poligoni vengono tracciati usando il line-mode del blitter. Un primo esempio molto semplice è presentato nel listato `lezione10o.s`, nel quale sono illustrati i vari tipi di fill. Quando l'area da riempire è delimitata da linee aventi pendenza minore di 45 gradi, sorge un problema. In questo caso, infatti, accade che una linea è formata da pixel che possono trovarsi adiacenti sulla stessa riga orizzontale dello schermo. La situazione è mostrata dalla seguente figura nella quale gli asterischi (*) rappresentano dei pixel di valore 1.



Come vedete quando una linea ha pendenza maggiore di 45 gradi non capita mai che 2 dei suoi pixel siano affiancati sulla stessa riga dello schermo. Al contrario ciò accade quando la pendenza della linea è minore di 45 gradi. Questo fatto crea il problema nel riempimento. Infatti quando il blitter incontra 2 pixel affiancati sulla stessa riga durante il riempimento, li considera come 2 bordi distinti, e quindi non riempie i pixel che si trovano a destra della linea. trovate un esempio di questo problema nel listato `lezione10p.s`. Per ovviare a questo problema, i progettisti del blitter ci hanno messo a disposizione una speciale modalità di tracciamento linee (a cui avevamo accennato in precedenza) che produce linee aventi un solo pixel per ogni riga orizzontale. Chiaramente se tracciate una linea in questa modalità senza poi fare il fill, essa vi apparirà "spezzettata". Nel listato `lezione10q.s` trovate la soluzione al problema mostrato in `lezione10p.s`. Nell'esempio `lezione10r.s` proviamo a tracciare e riempire un poligono chiuso formato da molte linee. Notiamo che si riscontra anche qui in piccolo problema. Il problema nasce dal fatto che i vertici del poligono sono in comune ad una coppia di linee. Quando disegniamo delle linee in modalità EOR invertiamo i pixel dello sfondo. I vertici vengono invertiti 2 volte e quindi alla fine risultano azzerati. Quindi c'è un "buco" nel bordo del poligono, a causa del quale il riempimento viene effettuato male. Se invece disegniamo le linee in modalità OR, i vertici rimangono al valore 1. Ciò crea problemi con i vertici in alto e in basso, in quanto essi si trovano isolati sulla riga a cui appartengono e pertanto il riempimento inizia a partire da essi ma non termina mai. Per capire meglio osservate la seguente figura (riferita al vertice basso):

```
       *        *
        *     *                Prima del FILL
         *  *
          *

          ^
          +---- vertice in basso


       **********
        *******                 Dopo del FILL
         ****
***********

          ^
          +---- vertice in basso
```

Come vedete sulla riga su cui giace l'ultimo vertice il riempimento non termina perché non c'è un altro pixel settato a 1 che faccia da bordo sinistro. Nel caso delle linee in modalità EOR questo problema non si presenta perché il vertice viene azzerato (cioè per il fenomeno che ci crea problemi per i vertici intermedi). Insomma in qualsiasi modo facciamo c'è sempre un vertice che ci fa sballare il fill!

Vediamo come trarci d'impaccio. Conviene disegnare le linee in modalità EOR, in modo da eliminare il problema dei vertici alto e basso. Inoltre facciamo in modo di disegnare le linee sempre dall'alto verso il basso e, prima di disegnarle invertiamo (con una BCHG) il primo pixel. In questo modo tale pixel sarà invertito 2 volte (dalla BCHG e poi dalla blittata) e risulterà dunque immutato. In questo modo il problema è risolto. Infatti (siccome abbiamo ordinato i punti) ogni vertice intermedio è disegnato una volta come ultimo pixel di una linea (e pertanto viene settato a 1) e una volta come primo pixel dell'altra linea (e pertanto rimane immutato, quindi a 1). Questa tecnica è presentata nell'esempio `lezione10s.s`.

Torniamo ora ad occuparci del trattamento di linee, per illustrare una particolarità. È possibile tracciare linee larghe 2 pixel semplicemente cambiando il valore di inizializzazione di `BLTBDAT`. La tecnica è illustrata nell'esempio `lezione10t1.s`. Nell'esempio `lezione10t2.s`, invece viene presentata una routine di tracciamento linee migliore di quella usata finora. Questa routine infatti sfrutta molti particolarità dell'assembler 68000 per ottimizzare il calcolo e il caricamento dei registri del blitter.

Per concludere la lezione presentiamo alcuni effetti realizzati mediante tracciamento di linee e fill, nei listati `lezione10u1.s`, `lezione10u2.s`, `lezione10v.s`, `lezione10x.s`. In particolare nell'ultimo vedrete una delle tecniche principali della leggendaria demo *"State of the Art"*!!

CAPITOLO 11

LEZIONE 11 - INTERRUPT, CIAA/CIAB, DOSLIB

(Directory `Sorgenti7`) - quindi scrivere *<V Assembler3:sorgenti7>*

Ora che avete approfondito il funzionamento del blitter, potete affermare con sicurezza di conoscere l'hardware di Amiga, dato che 68000, blitter e copper li sapete programmare. Però non si finisce mai di imparare, e in questa lezione vedremo delle informazioni avanzate sul 68000, come gli interrupt, nonché dei listati che mostrano utilizzi particolari di Blitter e Copper, e infine l'utilizzo dei chip CIAA e CIAB, che per ora abbiamo usato solo per testare la pressione del mouse. Direi di cominciare con le nuove informazioni sul 68000, in modo da fare una startup migliore di quella che abbiamo fatto nel corso della LEZIONE8. Le informazioni su vettori di eccezione, interrupt ecc. che saranno spiegate non saranno tutte utili e indispensabili per la programmazione di giochi e demo, ma solo una parte di esse ci servirà. Non lasciatevi dunque intimorire dalla quantità di cose accennate: in pratica è poco ciò che useremo! Innanzitutto occorre parlare dei due modi operativi del 680x0, ossia del modo utente e del modo supervisore. Già nel `68000-2.txt` si accennava, senza spiegare, che ci sono delle istruzioni "privilegiate", che devono essere eseguite in modo supervisore, ossia durante una eccezione o un interrupt. Per ora abbiamo fatto eseguire le nostre routines sempre in modo USER, ossia nel modo utente, perché non era necessario eseguire istruzioni privilegiate, e perché non abbiamo sfruttato gli Interrupt del processore.

Devo premettere che la velocità di esecuzione non cambia tra modo utente e modo supervisore, per cui far eseguire il proprio programma come una exception o un interrupt non turbizzerà di certo l'esecuzione. D'altronde una "eccezione" è chiamata così proprio perché si tratta di una cosa che deve avvenire solo in casi "eccezionali", per esempio un software failure, noto come GURU. Da notare che ci sono dei guru che sono causati dal sistema operativo Amiga, (come errori di exec) e altri che sono programmati direttamente dalla Motorola, autrice dei 680x0. Per esempio, gli errori di *bus*, di *divisione per zero*, ecc, ossia i vettori, sono di questo tipo. Avrete notato che il sistema operativo riesce a far aggiornare la posizione della freccia puntatore del mouse anche se stiamo eseguendo una nostra routine, a patto che non si disabiliti il multitasking con una chiamata al `Disable()`. Ebbene, se il processore sta eseguendo un nostro loop, come fa ogni fotogramma ad aggiornare altre cose? Ricorderete che se si disabilitano gli interrupt di

sistema si blocca tutto. Infatti si serve degli interrupt, che "interrompono" l'esecuzione del nostro programmino ogni fotogramma, eseguono una loro routine, e riprendono l'esecuzione del nostro programma dove l'avevano lasciata, il tutto senza che ce ne accorgiamo! Oltre agli interrupt, ci sono altri modi per eseguire delle routines in supervisore, ad esempio le istruzioni TRAP, o un qualsiasi errore in un programma. Ad esempio quando avviene una divisione per zero, o il processore trova dei dati che non corrispondono a nessuna istruzione, vengono eseguite in modo supervisore le routines dei guru meditation e software failure. Gli emulatori di altri processori, per esempio, fanno in modo che ad ogni valore binario di istruzione del processore emulato, ad esempio un 80286 o il 6502 del commodore 64, corrisponda una routine che faccia le operazioni che avrebbe fatto quell'istruzione per quel processore (grossomodo!). Ora, non è negli scopi di questo corso imparare a programmare sistemi operativi o emulatori, dato che il sistema operativo dell'Amiga è già il migliore al mondo, e gli emulatori su Amiga sono insuperabili, tanto che chi ha un Amiga può far eseguire i programmi di un MacIntosh e di un MSDOS, e giocare ai giochi dei vecchi C64 e Spectrum. Ma anche se si vuole programmare un demo o un gioco, o una utility come il protracker/octamed, è **utilissimo** gestire qualche interrupt.

Molte demo e giochi di vecchio tipo cominciavano mandando in modo supervisore il 68000, scrivendo subito sullo SR e facendo giochi strani con lo Stack. Ebbene, molti di questi giochi non funzionano su computer con 68020, perché lo Status Register nei processori più evoluti ha delle funzioni in più, che tali programmatori ignoravano, e settando o azzerando bit alla rinfusa commisero un enorme sbaglio. Inoltre in modo supervisore lo stack (SP) è uno stack "privato" del modo supervisore, mentre si accede allo stack di utente con il registro USP (User Stack Pointer). Insomma in modo supervisore è meglio andarci coi piedi di piombo, a meno che non si conoscano perfettamente tutte le nuove caratteristiche del 68020/30/40 e anche del 68060! Infatti il modo utente serve proprio a evitare di eseguire istruzioni che potrebbero provocare effetti diversi su processori diversi, che solo i sistemi operativi devono eseguire. Però i coder si sono sempre sentiti più tosti a mettere sottosopra i registri in modo supervisore, col risultato di farsi la fama di "programmatori incapaci di codice non compatibile".

Ma vediamo di eseguire qualche istruzione in modo supervisore, prima di continuare con la teoria. Tra i tanti modi che ci sono per far eseguire una routine in exception, il più "sicuro" è quello di usare l'apposita funzione del sistema operativo `exec.library/Supervisor`, che richiede in entrata di mettere in A5 l'indirizzo della routine da eseguire:

```
1         move.l  4.w,a6                  ; ExecBase in a6
2         lea     SuperCode(PC),a5        ; Routine da eseguire in supervisor
3         jsr     -$1e(a6)                ; LvoSupervisor - esegui la routine
4                                         ; (non salva i registri! attenzione!)
5         rts                             ; esci, dopo aver eseguito la routine
6                                         ; "SuperCode" in supervisor.
7
8 SuperCode:
9         movem.l d0-d7/a0-a6,-(SP)       ; Salva i registri nello stack
10        ...                             ; istruzioni da eseguire
11        ...                             ; come fosse una subroutine....
12        movem.l (SP),d0-d7/a0-a6        ; Riprende i registri dallo stack
13        RTE     ; Return From Exception: come l'RTS, ma per le eccezioni.
```

Come si può notare, non c'è niente di più facile. La routine da eseguire la potete considerare come una subroutine da chiamare con JSR o BSR, solo che si chiama con `JSR -$1e(a6)` dopo aver messo il suo indirizzo in A5, e naturalmente l'ExecBase in A6. Al termine della breve "subroutine supervisor", anziché mettere un RTS per ritornare, occorrerà mettere un RTE, apposito per tornare da eccezioni e interrupt. A questo punto il processore tornerà ad eseguire in modo utente l'istruzione sotto il `JSR -$1e(a6)`, proprio come fosse stato un BSR o un JSR. La funzione Supervisor non salva i registri nè li ripristina col MOVEM, per cui se se ne modifica qualcuno durante la routine in supervisor tali valori rimarranno al ritorno da quella routine. A questo proposito vi consiglio di salvare e ripristinare i registri manualmente all'inizio e alla fine della

routine supervisor. Da notare che accedendo a SP o A7 in modo supervisor si salva nello stack di supervisore, e non in quello utente, richiamabile come USP. Questo non è un problema, perché lo stack di supervisore salva e ripristina come quello utente, ma in caso di programmazione pericolosa dello stack potreste prendere dei granchi con impiantamento generale. Eseguiamo delle istruzioni privilegiate nell'esempio `lezione11a.s`. Ecco le istruzioni privilegiate eseguibili solo in modo supervisore:

```
1         ANDI.W  #xxxx,SR
2         ORI.W   #xxxx,SR
3         EORI.W  #xxxx,SR
4         MOVE.W  xxxxx,SR
5         MOVE.W  SR,xxxxx
6         MOVEC   registro,registro       ; 68010+ - registri speciali per
7                                         ; controllo cache, MMU, vettori come:
8                                         ; CAAR,CACR,DFC,ISP,MSP,SFC,USP,VBR.
9         RTE
```

Ci sarebbero anche MOVES, RESET, STOP, ma non ci interessano. Interessa poco la possibilità di agire sullo Status Register, perché è molto pericoloso (data la differenza dei bit dello SR tra 68000 e altri 680x0), e non è indispensabile disturbarlo. Tra l'altro quando si salta ad una eccezione viene salvato nello stack, oltre al Program Counter, anche lo Status Register, e al momento dell'RTE viene ripristinato il vecchio valore del Program Counter, per tornare sotto il `JSR -$1e(a6)`, e il vecchio SR, cosicché non ci sono cambiamenti, se non durante l'esecuzione dell'exception. Invece l'istruzione MOVEC ci interesserà molto di più, perché per poter usare un interrupt in modo che funzioni anche su processori 68020+ è necessario sapere dove si trova il VBR (Vector Base Register). Per quanto riguarda il controllo delle CACHE, credo sia meglio non interferire, in modo che l'utente possa attivarle o disattivarle con delle utility prima di eseguire il nostro demo o gioco, in modo da valutare le differenze di settaggio. Se invece decidiamo per lui quali cache devono essere attivate e quali disattivate, rischiamo anche di fare codice che non funziona sul 68060 e eventuali nuovi processori RISC che potrebbero emularlo. L'istruzione MOVEC, disponibile solo dal 68010 in avanti, serve per copiare il contenuto di un registro An o Dx in un registro speciale, o viceversa. Vediamo i registri speciali presenti nel 68020:

```
CAAR    - CAche Address Register
CACR    - CAche Control Register
VBR     - Vector Base register
```

Ci sono anche DFC (Destination Function Code), SFC (Source Function Code), ISP (Interrupt Stack Pointer), MSP (Master Stack Pointer), USP (User Stack Pointer), ma non ci interessano, perché servono solo a chi programma sistemi operativi, emulatori eccetera. A noi in questo momento serve sapere il valore del registro speciale VBR, vedremo poi il perché. Per ottenerlo basta un `MOVEC VBR,d0`, ad esempio. Ma cosa è il Vector Base Register? Innanzitutto occorre spiegare cosa è un vettore (da non confondere con i vettori in matematica, o i vectors 3d!). Prima vediamo la tabella dei vettori, poi spieghiamoli:

```
NUM. VETTORE   OFFSET   Assegnazione e significato

        0       $0      Serve solo al momento del reset (SSP start)
        1       $4      Serve solo al reset, infatti c'è ExecBase (PC start)
        2       $8      GURU/soft. failure: Errore di BUS
        3       $c      GURU/soft. failure: Errore di Indirizzo
        4       $10     GURU/soft. failure: Istruzione illegale
        5       $14     GURU/soft. failure: Divisione per zero
        6       $18     Eccezioni generate da istruzioni CHK,CHK2 (68020+)
        7       $1c     Eccezioni gen. da istruzioni TRAPV (68020+ TRAPCC)
        8       $20     GURU/soft. failure: Violazione di privilegio
        9       $24     Traccia (trace exception)
```

```
        $A      $28     GURU/soft. failure: Emulatore di linea %1010 (LINE-A)
        $B      $2c     GURU/soft. failure: Emulatore di linea %1111 (LINE-F)
        $C      $30     non usato
        $D      $34     Violazione di protocollo coprocessore (68020+)
        $E      $38     Errore di formato (solo 68020, dopo CALLM,RTM)
        $F      $3c     Interruzione non inizializzata
        ...     ...

        $18     $60     Interruzione spuria

; Ecco gli autovettori di interruzione: sono questi quelli che ci interessano!

        $19     $64     INTERRUPT livello 1 (softint,dskblk,tbe)
        $1a     $68     INTERRUPT livello 2 (ports: I/O,ciaa,int2)
        $1b     $6c     INTERRUPT livello 3 (coper,vblanc,blit)
        $1c     $70     INTERRUPT livello 4 (canali audio aud0/aud1/aud2/aud3)
        $1d     $74     INTERRUPT livello 5 (rbf,dsksync)
        $1e     $78     INTERRUPT livello 6 (exter: ciab,int6 + inten)
        $1f     $7c     INTERRUPT livello 7 (schede hardware esterne: NMI)

        $20     $80     Vettore richiamabile con TRAP #0
        $21     $84     Vettore richiamabile con TRAP #1
        $22     $88     Vettore richiamabile con TRAP #2
        $23     $8c     Vettore richiamabile con TRAP #3
        $24     $90     Vettore richiamabile con TRAP #4
        $25     $84     Vettore di TRAP #5, eccetera, fino TRAP #15
        ...

        seguono vettori per errori dell'eventuale coprocessore matematico
        e dell'MMU, che non ci interessano.
```

Immaginiamo la situazione di un processore 68000, per il quale non occorre cercare il valore del registro VBR, (non esiste nemmeno!). In questo caso l'offset è proprio la locazione di memoria! $0 = $00000000 All'indirizzo $4 troviamo l'Execbase, mentre la prima long a $0 di solito è azzerata. Ma ecco che "dentro" la locazione $8 troviamo l'indirizzo della routine che fa apparire la scritta *GURU MEDITATION/SOFTWARE FAILURE* in caso di *bus error*. Infatti tale guru, come visto nel 68000-2.TXT, ha come numero di identificazione #00000002, ossia il suo numero di vettore.

Il terzo vettore contiene l'indirizzo della routine che fa apparire il guru di ADDRESS ERROR (#00000003), e così via. In pratica, quando il processore trova uno di questi errori, salta al vettore corrispondente, che contiene l'indirizzo della routine da eseguire in modo supervisore (data la gravità della situazione!). Al momento del reset vengono scritti dalla ROM tutti gli indirizzi nei vettori dal primo all'ultimo. Se avete dei programmini che modificano i messaggi dei software failure o dei guru, sappiate che fanno "puntare" i vettori alle sue routines, anziché alle routines normali di sistema. Naturalmente ci sono modi "legali" per cambiare i vettori, ossia passando da strutture e routines del sistema operativo. Scrivere brutalmente l'indirizzo della propria routine nel vettore può risultare inefficace o incompatibile. Ad esempio:

```
1       MOVE.L  #MiaDivisionePerZero,$14.w       ; sostituisco il vettore
2                                                ; del guru da div. per zero.
3       rts
4
5   MiaDivisionePerZero:
6       ...
7       RTE
```

NON FATE MAI COME IN QUESTO ESEMPIO, PER UN PAIO DI RAGIONI. LA PRIMA è CHE NEL 68020+ NON è DETTO CHE TALE VETTORE SI TROVI ALL'INDIRIZZO $14, LA SECONDA è CHE è UN METODO INCOMPATIBILE CON MMU E STRUTTURE DEL SISTEMA OPERATIVO

AMIGA. Comunque, in teoria questo sistema dovrebbe funzionare, e su Amiga500 funziona quasi sempre, a patto che la routine supervisor sia scritta bene. La modifica dei vettori di errore/guru però ci interessa minimamente, perché il nostro programma non dovrebbe contenere errori da correggere al momento del salto al guru/software failure! Anche i vettori delle istruzioni `TRAP #xx` ci interessano poco. Tali vettori erano usati in passato per andare in eccezione, ma abbiamo già visto un modo più sicuro, quello tramite sistema operativo. Comunque, per vostra curiosità, il modo "antico" era:

```
        move.l  $80.w,OldVector         ; Salva il vecchio vettore TRAP #0
        move.l  #SuperCode,$80.w        ; Routine da eseguire in supervisor
                                        ; posta nel vettore TRAP #0
        TRAP    #0                      ; Esegui Supercode come eccezione
        move.l  OldVector(PC),$80.w     ; ripristina il vecchio vettore TRAP #0
        rts                             ; esci, dopo aver eseguito la routine
                                        ; "SuperCode" in supervisor.
OldVector:
        dc.l    0

SuperCode:
        movem.l d0-d7/a0-a6,-(SP)       ; Salva i registri nello stack
        ...                             ; istruzioni da eseguire
        ...                             ; come fosse una subroutine....
        movem.l (SP),d0-d7/a0-a6        ; Riprende i registri dallo stack
        RTE     ; Return From Exception: come l'RTS, ma per le eccezioni.
```

Come vedete si può facilmente capire la funzione dell'istruzione `TRAP`: in pratica se si esegue un `TRAP #0` viene eseguita come eccezione la routine il cui indirizzo è contenuto nella .l all'indirizzo $80, mentre con l'istruzione `TRAP #1` si esegue quella in $84, e così via. Allo stesso modo, gli interrupt contengono l'indirizzo della routine da eseguire in caso di interrupt. Il modo antico per settare un interrupt era:

```
        move.l  $6c.w,OldInt6c          ; Salva il vecchio int livello 3
        move.l  #MioInt6c,$6c.w         ; Mia routine per int livello 3
```

Alla fine del programma veniva ripristinato il vecchio interrupt in $6c. In questo interrupt di solito viene messo il `BSR.w MT_MUSIC`, dato che tale interrupt (`VERTB`) viene eseguito una volta per fotogramma.

11.1 Il registro VBR nei processori 68010 e superiori

Vi starete domandando cosa c'entra il VBR con i vettori. Ebbene, il Vector Base Register è l'indirizzo di BASE a cui aggiungere gli offset per trovare l'indirizzo dei vettori. Se il `VBR=0`, allora l'interrupt livello 3 si troverà all'indirizzo $6c, come nel 68000, e allo stesso modo il `TRAP #0` si troverà sempre in $80, e gli esempi visti sopra funzionerebbero. Ma se il `VBR` fosse a $10000, l'interrupt di livello 3 si troverebbe non più a $6c, ma a `VBR`+$6c, ossia a $1006c! Lo stesso per tutti gli altri vettori. Dunque, in linea di massima:

```
indirizzo del vettore = VBR + OFFSET
```

Sul processore 68000 la base è sempre $0000, tanto che non esiste il registro `VBR` né l'istruzione privilegiata `MOVEC`. Ma su 68010 e superiori il `VBR` può essere spostato in altri luoghi, anche in FAST RAM. Dopo un reset, il `VBR` è comunque sempre azzerato, sia su A3000 che su A1200 o A4000. È eseguendo il `SETPATCH` o altre utility che il `VBR` viene spostato. Infatti, molte demo/giochi su file che funzionano se fatte partire da sole da un disco senza caricare prima il setpatch, non funzionano se caricate dallo shell del workbench. Alcune funzionano ma sono "mute", proprio perché scrivono il loro interrupt, che suona solo la musica, in $6c, quando invece il VBR punta più avanti di $0000. Dato che sappiamo questo, basta controllare se il processore è un 68000 o

un 68010+, e nel caso sia un 68010 o superiore prelevare il valore del VBR e sommarlo al vettore che si vuole cambiare. Ecco come fare in pratica:

```
        move.l  4.w,a6          ; ExecBase in a6
        btst.b  #0,$129(a6)     ; Testa se siamo su un 68010 o superiore
        beq.s   IntOK           ; è un 68000! Allora la base è sempre zero.
        lea     SuperCode(PC),a5 ; Routine da eseguire in supervisor
        jsr     -$1e(a6)        ; LvoSupervisor - esegui la routine
                                ; (non salva i registri! attenzione!)
        bra.s   IntOK           ; Abbiamo il valore del VBR, continuiamo...

;**********************CODICE IN SUPERVISORE per 68010+ **********************
SuperCode:
        movem.l a0-a1,-(SP)     ; Salvo a0 ed a1 nello stack
        dc.l    $4e7a9801       ; Movec Vbr,A1 (istruzione 68010+).
                                ; è in esadecimale perchè non tutti gli
                                ; assemblatori assemblano il movec.
        lea     BaseVBR(PC),a0  ; Label dove salvare il valore del VBR
        move.l  a1,(a0)         ; Salvo il valore.
        movem.l (SP)+,a0-a1     ; Ripristino i vecchi valori di a0 ed a1
        RTE                     ; Ritorna dalla eccezione
;*****************************************************************************

BaseVBR:
        dc.l    0

IntOK:
        move.l  BaseVBR(PC),a0   ; In a0 il valore del VBR
        move.l  $64(a0),OldInt64 ; Sys int liv 1 salvato (softint,dskblk)
        move.l  $68(a0),OldInt68 ; Sys int liv 2 salvato (I/O,ciaa,int2)
        move.l  $6c(a0),OldInt6c ; Sys int liv 3 salvato (coper,vblanc,blit)
        move.l  $70(a0),OldInt70 ; Sys int liv 4 salvato (audio)
        move.l  $74(a0),OldInt74 ; Sys int liv 5 salvato (rbf,dsksync)
        move.l  $78(a0),OldInt78 ; Sys int liv 6 salvato (exter,ciab,inten)

        movem.l d0-d7/a0-a6,-(Sp)       ; Salva i registri nello stack
        bsr.s   START                   ; Esegui la routine principale
        movem.l (sp)+,d0-d7/a0-a6       ; Riprendi i registri dallo stack

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  OldInt64(PC),$64(a0) ; Sys int liv1 salvato (softint,dskblk)
        move.l  OldInt68(PC),$68(a0) ; Sys int liv2 salvato (I/O,ciaa,int2)
        move.l  OldInt6c(PC),$6c(a0) ; Sys int liv3 salvato (coper,vblanc,blit)
        move.l  OldInt70(PC),$70(a0) ; Sys int liv4 salvato (audio)
        move.l  OldInt74(PC),$74(a0) ; Sys int liv5 salvato (rbf,dsksync)
        move.l  OldInt78(PC),$78(a0) ; Sys int liv6 salvato (exter,ciab,inten)
        rts

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        ...
        eseguo il programma
        ...
        rts
```

Da notare che anche se si volesse usare l'istruzione TRAP occorrerebbe mettere in a0 il BaseVbr e fare l'offset $80(a0). Lo stesso dicasi per tutti i vettori. è presente da questa lezione in avanti una startup nuova, startup2.s, da includere al posto della startup1.s. L'unica differenza è che contiene le istruzioni viste sopra ed è disponibile la label BaseVbr per modificare gli interrupt propriamente su tutti i microprocessori. Il salvataggio dei vecchi interrupt ed il ripristino alla fine sono fatti dalla startup, assieme al salvataggio e al ripristino dei canali DMA e di INTENA. Un'altra modifica è l'aggiunta di una routine che blocca l'input del mouse e della tastiera al sistema operativo, vedremo che caricando i file servirà. Ora che sappiamo come sostituire un interrupt di sistema con uno nostro, dobbiamo vedere come fare il nostro interrupt. Per anticipare le pagine seguenti, suoniamo la musica in interrupt nel listato esempio Lezione11b.s. Capirete meglio come funziona continuando a leggere!

11.2 Come si costruisce una routine di interrupt

Il sistema delle interruzioni (interrupt) consente ad un dispositivo esterno o ad un chip custom di interrompere l'esecuzione del processore, per farlo saltare in modo user alla routine il cui indirizzo si deve trovare in uno degli autovettori di interruzione (ad esempio $6c). Questi interrupt hanno dei livelli diversi di *priorità*, che vanno da un livello minimo (1) al livello massimo (7). Queste priorità servono nel caso che si presenti l'occorrenza di una o più interruzioni durante l'esecuzione di un interrupt. Se per esempio il programma normale in modo user viene interrotto da un interrupt di livello basso, diciamo 2, e mentre viene eseguita questa routine si presenta la richiesta di un interrupt di livello più alto, ad esempio 5, l'interrupt 2 viene interrotto a sua volta dall'interrupt 5, con maggiore priorità, e una volta terminato quest'ultimo interrupt il controllo torna all'interrupt di livello 2, il quale poi lo ripassa al programma normale in modo user. In questo modo più routines interrupt possono rimanere in attesa di terminare l'esecuzione, e a seconda del loro livello saranno finite di eseguire prima. La necessità di introdurre gli interrupt fin dai primi microprocessori è legata al fatto che spesso si utilizza male la potenza della CPU, a causa di loop di attesa molto lunghi. Se per esempio si dovesse attendere l'inizio del vertical blank, si dovrebbe fare un loop che controlla la linea raggiunta, e fino a che tale linea non è raggiunta il processore non fa altro che bloccarsi in quel ridicolo loop. Se l'attesa per un certo segnale fosse di qualche secondo, immaginatevi come sarebbe sprecata la potenza del processore! Per questo esistono gli interrupt, che permettono al processore di eseguire dei programmi senza preoccuparsi di attendere gli "eventi". Se si genera un interrupt di livello 3 ogni vertical blank, si potrà far calcolare al processore un frattale o una immagine 3d, e quando si presenterà l'interrupt, al momento del vertical blanking, sarà interrotto il calcolo del solido 3d, verrà eseguita la routine da eseguire all'inizio del Vblank, poi si tornerà a continuare la routine 3d dove si era lasciata. Il multitasking stesso è possibile grazie a questo: poter stampare o leggere dal disk drive mentre si fa qualcos'altro è possibile perché, a differenza del PC MSDOS, il processore può svolgere un compito che sarà interrotto al momento giusto dall'interrupt del disco o della porta seriale/parallela, e che sarà ripreso appena eseguiti tali interrupt. L'Amiga assegna 6 dei 7 livelli di interrupt disponibili, a segnali prodotti dai chip custom (blitter, copper, cia) in determinate situazioni. Il settimo livello è usato da schede esterne come la Action Replay, dato che le linee IPL2-IPL0 che lo generano sono portate alla porta di espansione. I primi 6 livelli di interrupt sono generati dai chip custom, ad esempio in occasione del completamento di una blittata o di un vertical blank video.

11.3 Come si usano INTENA ed INTENAR

È possibile, tramite il registro INTENA ($dff09a), mascherare alcuni di questi interrupt, ossia evitare che siano generati. Esiste anche un registro per richiedere interrupt, l'INTREQ ($dff09c). Questi registri funzionano come il DMACON ($dff09a), infatti il bit 15 decide se gli altri bit specificati devono essere settati o azzerati. Come abbiamo fatto per il DMACON/DMACONR nella lezione 8, vediamo la "mappa" dei registri INTENA($dff09a) a sola scrittura e INTENAR($dff01c) a sola lettura:

```
INTENA/INTENAR ($dff09a/$dff01c)

BIT     NOME     LIV.   DESCRIZIONE

15      SET/CLR         Bit di controllo "Set/clear". Determina se i bit ad 1
                        devono essere azzerati o settati, come in DMACON.
                        I bit=0 non saranno nè settati nèazzerati
14      INTEN           Master interrupt (interrutt. generale di abilitazione)
```

```
13      EXTER   6 ($78) Interrupt esterno, connesso alla linea INT6
12      DSKSYN  5 ($74) Generato se il registro DSKSYNC corrisponde ai dati
                        letti dal disco nel drive.Serve per i loader hardware.
11      RBF     5 ($74) Buffer UART di ricezione della porta seriale PIENO.
10      AUD3    4 ($70) Lettura di un blocco di dati del can. audio 3 finita.
09      AUD2    4 ($70) Lettura di un blocco di dati del can. audio 2 finita.
08      AUD1    4 ($70) Lettura di un blocco di dati del can. audio 1 finita.
07      AUD0    4 ($70) Lettura di un blocco di dati del can. audio 0 finita.
06      BLIT    3 ($6c) Se il blitter ha finito una blittata si setta ad 1
05      VERTB   3 ($6c) Generato ogni volta che il pennello elettronico è
                        alla linea 00, ossia ad ogni inizio di vertical blank.
04      COPER   3 ($6c) Si può settare col copper per generarlo ad una certa
                        linea video. Basta richiederlo dopo un certo WAIT.
03      PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2
02      SOFT    1 ($64) Riservato agli interrupt inizializzati via software.
01      DSKBLK  1 ($64) Fine del trasferimento di un blocco dati dal disco.
00      TBE     1 ($64) Buffer UART di trasmissione della porta seriale VUOTO.
```

Come si può notare, l'analogia con DMACON/DMACONR è evidente:

- Il bit 15 è importantissimo: se esso è acceso allora i bit settati a 1 in scrittura nel $dff09A servono ad abilitare i relativi interrupt, se il bit 15 è a 0, allora gli altri bit a 1 nel registro servono a disabilitare, ossia a mascherare, i relativi interrupt. Per abilitare o disabilitare uno o più interrupt, come in DMACON, è comunque necessario impostare ad 1 i relativi bit; quello che determina se quegli interrupt devono venir abilitati o disabilitati è il bit 15: se è ad 1 si abilitano, mentre se è a 0 si spengono (sempre indipendentemente dal loro precedente stato). Diciamo che si sceglie su quali bit OPERARE, poi si decide se attivarli(0) o disattivarli(1) in base al bit 15. I bit 0 non sono nè settati nè azzerati.

  Facciamo un esempio:

```
1                ;5432109876543210
2       move.w #%1000000111000000,$dff09A ; sono ABILITATI i bits 6,7 e 8
3                ;5432109876543210
4       move.w #%0000000100100000,$dff09A ; sono DISABILITATI i bits 5 e 8.
```

- Il bit 14 funge da interruttore generale (come fa il bit 9 nel DMACON). Lo si può azzerare ad esempio per disabilitare momentaneamente tutti i livelli di interrupt, senza ricorrere all'azzeramento dell'intero registro.

Vi ricorderete della vecchia Lezione3a.s che con un:

```
1       MOVE.W  #$4000,$dff09a  ; INTENA - Ferma gli interrupt
```

Si bloccavano tutti gli interrupt, mentre con:

```
1       MOVE.W  #$C000,$dff09a  ; INTENA - Riabilita gli interrupt
```

Si riabilitavano tutti. Ebbene, $4000 = %0100000000000000, ossia si azzera il bit MASTER, il 14. Invece $c000 = %1100000000000000, cioè si riabilita il bit MASTER, e con esso tutti gli interrupt. In Lezione11b.s per abilitare VERTB:

```
1       move.w  #$c020,$9a(a5)  ; INTENA - abilito interrupt "VERTB" del
2                               ; livello 3 ($6c), quello che viene generato
3                               ; una volta al fotogramma (alla linea $00).
4
5               ; 5432109876543210
6  Infatti, $c020 = %1100000000100000 - bit 5, VERTB, settato assieme al MASTER.
```

Come avrete notato, gli interrupt spaziano dalla gestione della porta seriale alla lettura della sincronia delle tracce del disk drive, senza risparmiare nè blitter nè CIA, nè COPPER. Ridefinirsi

tutti i livelli di interrupt è molto pericoloso dal punto di vista della compatibilità, ed è anche molto difficile e specifico di chi voglia farsi un proprio sistema operativo. Ai programmatori di demo e giochi interessa soltanto l'interrupt $6c, ossia quello di livello 3, che riguarda la generazione di interrupt sincronizzati col pennello elettronico (VERTB - bit 5), o generabili a particolari linee video con il copper (COPER - bit 4). Più raramente può succedere di dover avere a che fare con interrupt per la gestione della tastiera o altro. In particolare il caricamento da disco tramite loader hardware è fuori moda, perché occorre fare giochi e demo installabili su hard disk, per cui gli interrupt del disk drive non ci interesseranno. Inoltre, anche se voleste fare un gioco che sfrutti la porta seriale per giocare in doppio su 2 computer collegati via cavo o via modem, sarebbe meglio usare le chiamate legali da sistema operativo del SERIAL.DEVICE, anziché farsi degli interrupt poco compatibili con eventuali schede multiseriali o nuovo hardware.

11.4 Come si usano INTREQ e INTREQR

In Lezione11b.s abbiamo visto anche INTREQ/INTREQR. Di cosa si tratta? Avrete notato come è strutturato l'interrupt $6c:

```
1  MioInt6c:
2          btst.b  #5,$dff01f      ; INTREQR — il bit 5, VERTB, è azzerato?
3          beq.s   NointVERTB              ; Se si, non è un "vero" int VERTB!
4          movem.l d0–d7/a0–a6,–(SP)       ; salvo i registri nello stack
5          bsr.w   mt_music                ; suono la musica
6          movem.l (SP)+,d0–d7/a0–a6       ; riprendo i reg. dallo stack
7  nointVERTB:      ;6543210
8          move.w  #%1110000,$dff09c ; INTREQ — cancello rich, BLIT,COPER,VERTB
9                                  ; dato che il 680x0 non la cancella da solo!!!
10         rte     ; uscita dall'int COPER/BLIT/VERTB
```

NOTA: INTREQR è la word $dff01e/1f. In questo caso agiamo sul suo byte $dff01f anziché su $dff01e, ma si tratta sempre del byte basso di INTREQR.

La mappa di INTREQ/INTREQR è uguale a quella di INTENA/INTENAR:

```
INTREQ/INTREQR ($dff09c/$dff01e)

BIT     NOME     LIV.   DESCRIZIONE

15      SET/CLR         Bit di controllo "Set/clear". Determina se i bit ad 1
                        devono essere azzerati o settati, come in DMACON.
                        I bit=0 non saranno nè settati nèazzerati
14      INTEN   6 ($78) interrupt livello 6 CIAB
13      EXTER   6 ($78) Interrupt esterno, connesso alla linea INT6
12      DSKSYN  5 ($74) Generato se il registro DSKSYNC corrisponde ai dati
                        letti dal disco nel drive.Serve per i loader hardware.
11      RBF     5 ($74) Buffer UART di ricezione della porta seriale PIENO.
10      AUD3    4 ($70) Lettura di un blocco di dati del can. audio 3 finita.
09      AUD2    4 ($70) Lettura di un blocco di dati del can. audio 2 finita.
08      AUD1    4 ($70) Lettura di un blocco di dati del can. audio 1 finita.
07      AUD0    4 ($70) Lettura di un blocco di dati del can. audio 0 finita.
06      BLIT    3 ($6c) Se il blitter ha finito una blittata si setta ad 1
05      VERTB   3 ($6c) Generato ogni volta che il pennello elettronico è
                        alla linea 00, ossia ad ogni inizio di vertical blank.
04      COPER   3 ($6c) Si può settare col copper per generarlo ad una certa
                        linea video. Basta richiederlo dopo un certo WAIT.
03      PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2
02      SOFT    1 ($64) Riservato agli interrupt inizializzati via software.
01      DSKBLK  1 ($64) Fine del trasferimento di un blocco dati dal disco.
00      TBE     1 ($64) Buffer UART di trasmissione della porta seriale VUOTO.
```

A cosa serve un registro per la richiesta di interrupt? A richiedere interrupt, naturalmente. E anche a **disdirli**, dato che una volta che viene richiesto un interrupt, automaticamente (dai chip custom) o manualmente (dal nostro programma), viene eseguito tale interrupt, ma non viene cancellata la "richiesta", per cui alla fine di ogni interrupt occorre cancellare gli interrupt già svolti dalla lista degli interrupt da fare.

L'INTREQ ($dff09c) è usato dal 680x0 per forzare l'esecuzione di un interrupt, di solito l'interrupt software, oppure dal COPPER per eseguire l'interrupt COPER ad una certa linea video. Naturalmente una volta settata una richiesta di interrupt, se tale interrupt non è abilitato in INTENA si può attendere anche una vita. Quando un bit settato in INTREQ è contemporaneamente settato anche in INTENA si verifica l'interrupt corrispondente a quel bit. Attenzione alla particolarità che se il bit 14 di INTREQ è settato si verifica un interrupt di livello 6 (a patto che il corrispondente bit in INTENA, Master Enable, sia anch'esso settato) Altrimenti è usato per cancellare i bit di richiesta degli interrupt già eseguiti, dato che le richieste di interrupt non sono azzerate automaticamente. Occorre stare attenti a questo fatto, perché se si dimentica di cancellare i bit di richiesta alla fine di ogni interrupt eseguito, il processore lo eseguirà nuovamente!! Ora dovreste capire la parte finale dell'interrupt:

```
1               ;6543210
2       move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,COPER,VERTB
3                                 ; dato che il 680x0 non la cancella da solo!!!
4       rte     ; uscita dall'int COPER/BLIT/VERTB
```

l'INTREQR ($dff01e) è a sola lettura, al contrario di INTREQ che è a sola scrittura. Serve per sapere quale chip ha richiesto l'interrupt. Infatti, se viene eseguito l'interrupt di livello 3 ($6c), può essere "colpa" del blitter, del vertical blank o del copper. Testando i bit di INTREQR capiamo quale di queste 3 è la causa, e determiniamo quale routine eseguire, o se eseguire la routine, nel caso ci interessi solo una di queste 3 eventualità. Il bit 15 non ha significati nell'INTREQR, dato che è il Set/Clr. Rivediamo ora l'utilizzo fatto in Lezione11b.s:

```
1       btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, è azzerato?
2       beq.s   NointVERTB      ; Se si, non è un "vero" int VERTB!
```

In questo caso, dato che il BTST su un indirizzo può essere solo .BYTE, è testato il $dff01f, ossia il byte basso della word, anziché il $dff01e. Nel caso che si salti a Noint, è evidente che l'interrupt è stato generato dal copper o dal blitter, e sarebbero settati i bit 4 o 6. A tal proposito occorre disdire anche queste richieste di interrupt, per evitare che ogni microsecondo si ritorni ad eseguire l'interrupt per nulla:

```
1               ;6543210
2       move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,COPER,VERTB
3                                 ; dato che il 680x0 non la cancella da solo!!!
4       rte     ; uscita dall'int COPER/BLIT/VERTB
```

Vi sembrerà strano che, nonostante sia abilitato con INTENA soltanto VERTB, possa succedere che vengano richiesti ed **eseguiti** interrupt di COPER o BLIT. Effettivamente non dovrebbero essere richiesti, nè eseguiti... Ma per motivi legati probabilmente alla MMU o alla velocità del processore, su computer più veloci del 1200 base, come l'A4000, può accadere benissimo, e ciò infatti causa dei problemi a delle demo, anche alcune recenti per AGA. Infatti su A1200 base può succedere che l'interrupt funzioni anche senza il BTST del bit VERTB, ma su A4000 o A1200 turbizzato non è così. Ammetto che "in teoria" dovrebbe funzionare, ma il fatto è che molte demo per A1200, se eseguite su A4000 suonano la musica 2 volte per fotogramma, e sono ridicole. Dunque siate categorici nel testare sempre i bit di INTREQ prima di eseguire l'interrupt, anche se sul vostro computer funziona tutto, o vi troverete con un gioco/demo che fa le bizze su a4000 e company.

Per riassumere, ecco le cose da fare per settare un nostro interrupt:

- Ottenere l'indirizzo del VBR, salvare il vecchio interrupt e ripristinarlo prima di uscire. Questo compito è svolto bene da `startup2.s`, non c'è problema: l'indirizzo del VBR è nella label `BaseVBR`.
- Azzerare tutti gli interrupt con INTENA. Anche questo compito è svolto dalla `startup2.s`, con un `MOVE.W #$7fff,$9a(a5)`.
- Mettere l'indirizzo del nostro interrupt nell'autovettore giusto.
- Abilitare solo l'interrupt, o gli interrupt che ci servono

Ed ecco cosa ricordarsi di mettere nella nostra routine interrupt:

- Salvare e ripristinare tutti i registri con un bel MOVEM, dato che se si "sporcasse" qualche registro, immaginatevi alla fine dell'interrupt cosa succederebbe, quando si torna ad eseguire un programma interrotto in chissà quale situazione e con chissà quali valori nei registri!
- Testare subito il $dff01e/1f (INTREQR), per sapere chi o che cosa ha generato un interrupt di quel livello. Per esempio, un interrupt di livello 3 può essere generato da COPER, VERTB o BLITTER; un interrupt di livello 4 da AUD0, AUD1, AUD2 o AUD3, eccetera. Attenzione al fatto che anche se alle volte sembra funzionare senza questo test, su A4000 o simili andrà tutto sfasato come se la CPU fosse ubriaca (può essere un effetto speciale, pero!)
- Cancellare i bit di INTREQ ($dff09c) che hanno causato l'interrupt eseguito, dato che non sono cancellati automaticamente. Se ci si dimentica di fare questo il processore avrà la richiesta fissa di interrupt e sarà eseguito continuamente.
- Terminare l'interrupt con un RTE, come si termina una subroutine con RTS.

Alla luce di queste considerazioni, vi ripropongo il primo interrupt:

```
1  MioInt6c:
2          btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, è azzerato?
3          beq.s   NointVERTB              ; Se si, non è un "vero" int VERTB!
4          movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
5          bsr.w   mt_music                ; suono la musica
6          movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
7  nointVERTB:      ;6543210
8          move.w  #%1110000,$dff09c ; INTREQ - cancello rich, BLIT,COPER,VERTB
9                                    ; dato che il 680x0 non la cancella da solo!!!
10         rte     ; uscita dall'int COPER/BLIT/VERTB
```

11.5 Gli interrupt e il sistema operativo

Il 680x0 ha solo 7 livelli di INTERRUPT, ma allora come è possibile che gli interrupt in pratica siano 15? Ebbene, il chip Paula si occupa di dividere in pseudointerrupt i 7 livelli "reali" di interrupt. Per esempio fa saltare all'interrupt di livello 3 in tre casi: COPER, VERTB e BLIT, e l'unico modo di sapere quale di queste tre eventualità ha generato l'interrupt è di consultare un registro collegato al chip Paula stesso, ossia INTREQR! D'altronde, essendo solo 7 i livelli di interrupt "reali" del 680x0, non è possibile che interrupt "sdoppiati" da Paula dello stesso livello possano interrompersi tra loro. Mentre un interrupt di livello 5, come il DSKSYNC, può interrompere l'esecuzione di un'interrupt di livello 3, come COPER, non è possibile per BLIT interrompere COPER, anche se ha "priorità Paulesca" maggiore, dato che si trovano nello stesso livello fisico del 680x0. Per questo se durante l'esecuzione di un interrupt si verifica la richiesta di interrupt di un altro pseudo-livello di Paula nello stesso livello 680x0, come ad esempio uno

BLIT mentre si esegue un COPER, al termine dell'int che esegue il COPER verrà eseguito subito un'altra volta l'interrupt di livello 3, questa volta eseguendo la routine per COPER (secondo il BTST fatto sull'INTREQR verrà identificato quale tipo di "sottoint" eseguire). Ecco le priorità dei livelli di interrupt nel sistema operativo, ossia nell'Exec.library, che come vedete segue la priorità hardware:

```
livello 1: ($64)        MINIMA PRIORITà

1       buffer di trasmissione vuoto    TBE
2       blocco del disco trasferito     DSKBLK
3       interrupt software              SOFTINT

livello2: ($68)

4       porte esterne INT2 & CIAA       PORTS

livello3: ($6c)

5       copper                          COPER
6       intervallo di vertical blank    VERTB
7       blittata finita                 BLIT

livello4: ($70)

8       canale audio 2                  AUD2
9       canale audio 0                  AUD0
10      canale audio 3                  AUD3
11      canale audio 1                  AUD1

livello5: ($74)

12      buffer di ricezione pieno       RBF
13      sync del disco trovata          DSKSYNC

livello6: ($78)         MASSIMA PRIORITà

14      external INT6 & CIAB            EXTER
15      speciale (abilitazione master)  INTEN

livello7: ($7c) (schede esterne come la Action Replay)

-       interrupt non mascherabile      NMI
```

Il fatto che il sistema operativo gestisca sue routines interrupt rende più pericolosa la sostituzione di alcuni di essi da parte nostra. Per quanto riguarda la priorità 6, la graphics.library usa l'interrupt del timer CIAB Time Of Day (TOD) per controllare lo schermo. Nella priorità 5, DSKSYNC è usato dal TrackDisk e RBF dal serial.device. Nel livello 4, ci sono i canali audio, usati dall'audio.device. Nel livello 3, l'interrupt BLIT, che avviene quando il blitter ha finito una sua operazione, spesso vengono messe routines che si occupano di riutilizzare i dati appena scritti dal blitter, per evitare di perdere tempo. Nel livello 2, del chip CIAA il Timer.device usa l'interrupt TimerA per l'handshake di tastiera, il TimerB per il timer a microsecondi, e l'interrupt TOD alarm a 50/60Hz. Esiste anche l'INT2 per eventuali schede hardware esterne. Nel livello 1, il più basso, l'interrupt TBE è usato dal Serial.device, l'interrupt DSKBLK è usato dal TrackDisk.device. Gli interrupt SOFTINT, ossia software, sono definibili via sistema operativo, ad esempio con la funzione Cause dell'Exec o facendo una message port di tipo SOFT_INT.

11.6 Gli interrupt COPER chiamati da COPPERList

Se si vuole chiamare l'interrupt COPER del livello 3 ($6c) ad una certa linea video, basta scrivere $8010 nell'intreq ($dff09c), dopo un wait che aspetti quella linea video:

```
COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU
        dc.w    $a007,$fffe     ; WAIT - attendi la linea $a0
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

Infatti il valore $8010 = $8000 + %10000, ossia si setta il bit 4, COPER. Vediamo in Lezione11c.s un esempio pratico.

Naturalmente l'interrupt si può chiamare anche a diverse linee, ogni volta cambiando "effetto". Verifichiamolo in Lezione11d.s.

Data la particolare complessità degli interrupt, per ora non faremo esempi ulteriori, riguardo ad interrupt dei dischi, della porta seriale ecc. Alle applicazioni che ci interessano, ossia DEMO e GIOCHI, bastano spesso solo i due tipi di interrupt di livello 3 ($6c) che abbiamo visto, ossia il VERTB, che viene eseguito ogni fotogramma, e il COPER, richiamabile dal copper a una qualsiasi linea video. Le applicazioni degli altri interrupt saranno commentate man mano che saranno trovate nei listati esempio, dato che spaziano in ogni campo! Per ora, possiamo anticipare l'uso di tutti i livelli di interrupt: nei listati Lezione11e.s e Lezione11f.s sono ridefiniti TUTTI gli interrupt, e sono abilitati TUTTI i livelli, ma naturalmente ci sono routines solo nel livello 3. Questo esempio può essere utile come "partenza" per definirsi un qualsiasi livello interrupt: potete "ritagliare" il livello che vi interessa e metterci le routines "dentro".

11.7 Informazioni avanzate sul COPPER - Uso del solo COLOR0 ($180) - No BitPlanes

Come recita il titolo, ora vedremo le applicazioni possibili usando solo una copperlist senza bitplanes. Ossia creare con soli WAIT e MOVE dei disegni o delle animazioni. Naturalmente, se aggiungete bitplanes potete "incrociare" e "sovrapporre" questi effetti, modificando molte volte per copperlist il color2 o il color3, oltre al color0. Per iniziare, però, occorre spiegare delle cose che non sono ancora state trattate. Si tratta del "tempo" impiegato dal copper per eseguire un suo comando MOVE. Abbiamo già visto come cambiare l'intera palette, anche di 32 colori, ad una certa linea video, per far comparire su schermo qualche centinaio di colori, col solo "32" o "16" colori ufficialmente settati nel BPLCON0. Ebbene, in una linea siamo riusciti a cambiare 32 colori, ossia ad eseguire 32 MOVE del copper:

```
        dc.w    $180,xxx        ; 1 move nel color0
        dc.w    $182,xxx        ; 1 move nel color1
        ...                     ; ecc.
```

Ebbene, in realtà se cominciamo a cambiare i colori dalla posizione orizzontale $07, o anche $01, tutti i 32 colori saranno effettivamente cambiati solamente verso la metà dello schermo, perché ogni MOVE richiede 8 pixel lowres per essere eseguito. Per questo conviene sempre cambiare i colori 1 linea prima che inizi veramente il disegno. Se mettessimo una cinquantina di MOVE di seguito, arriveremmo con l'ultima alla linea sotto! D'altronde, sarebbe materialmente impossibile eseguire decine di MOVE in meno di 1 cinquantesimo di cinquantesimo di secondo! Comunque possiamo usare questa apparente limitazione per i nostri scopi, ad esempio per cambiare colore in senso orizzontale ogni 8 pixel, senza usare i WAIT, ma semplicemente affiancando una cinquantina di COLOR0 per linea. Vediamo un esempio pratico di questo in Lezione11g1.s.

Un utilizzo di questo listato potrebbe essere quello di cambiare il $182, e non il $180, in uno schermo a 1 bitplane: in questo modo le eventuali "scritte" in sovraimpressione sarebbero sfumate da sinistra verso destra anziché dall'alto verso il basso, come facciamo di solito con la copperlist.

In `Lezione11g2.s` e `Lezione11g3.s` ci sono versioni più colorate di questo effetto, tanto che possono essere una base per degli effetti "PLASMA".

A proposito, se "roteassimo" o "ciclassimo" i colori di una linea di questo tipo cosa avremmo? Un effetto molto noto, usato dalle intro già negli albori dell'Amiga: vediamo l'effetto "supercar" in `Lezione11g4.s`.

Forse ciclare solo 2 linee non esalta. Vediamo di ciclarne di più, magari facendo un effetto di tipo "annodato", in `Lezione11g5.s`.

Un'altra "fantasia" che sfrutta l'"affollamento" di colorXX messi di seguito, nell'esempio `Lezione11g6.s`.

Ora vediamo un modo un pò "economico" per fare un simil-plasma: anziché cambiare i contenuti dei molti color0, mettiamo un wait all'inizio di ogni linea, ognuna delle quali conterrà 52 color0: possiamo spostare la linea a destra e a sinistra semplicemente cambiando la posizione orizzontale dei vari wait! In pratica questo è in `Lezione11g7.s`.

11.8 Uso della COPPER2 (COP2LC/COPJMP2)

Avrete notato che oltre al $dff080 e il $dff088 per puntare e far partire la copper 1, esistono il $dff084 e il $dff08a per puntare e far partire la copper2. Ma come funziona la copper2? E in che cosa può servirci? Ogni inizio frame il copper fa partire la copper 1, il cui indirizzo viene letto dal $dff080. Noi a volte lo facciamo partire "al volo", senza attendere nemmeno la fine del frame, scrivendo nel `COPJMP1`, ossia $dff088. Se mettessimo una copperlist nel $dff084, (`COP2LC`), dovremmo anche farla partire scrivendo nel `COPJMP2` ($dff08a). Ma alla fine del frame ripartirebbe la copper1. Ora, questa caratteristica può servire per fare diverse copperlist a cui saltare, come facciamo per le istruzioni 680x0 con i `JMP`. Per esempio, se volessimo eseguire fino a metà schermo la copper1, dopodiché saltare ad eseguire l'altra metà dalla copper 2, basterebbe puntare la copper 2 all'inizio, e farla partire dalla copperlist tramite `COPJMP2`:

```
1         move.l  #copper1,$dff080        ; COP1LC
2         move.l  #copper2,$dff084        ; COP2LC
3         ...
4
5         section copperissime,data_C
6
7   copper1:
8         ...     ; istruzioni varie...
9         dc.w    $a007,$fffe     ; Aspetta linea $a0
10        dc.w    $8a,0           ; COPJMP2 - fai partire la copper 2
11
12
13  copper2:
14        ...     ; istruzioni varie
15        dc.w    $ffff,$fffe     ; Fine della copperlist, si ripartirà con
16                                  ; la copper1!
```

Il copper, arrivato al `dc.w $8a,0` salta (come `BRA` o `JMP`) a copper2, a patto che questa fosse messa in $dff08a preventivamente. Da notare che SALTA, e non fa come un `BSR`, per cui non torna mai sotto il `dc.w $8a,0` della copper1. Vediamo ora un paio di utilizzi pratici della copper2. Uno è quello di fare le cosiddette copper dinamiche, ossia composte da 2 copperlist scambiate ogni fotogramma, come un "double buffering" dei bitplane. Questo serve per rendere più "lisce" le sfumature, infatti se scambiate 2 colori ogni fotogramma, avrete un effetto tipo l'interlace, che

farà "vedere" il colore intermedio. Basta prepararsi 2 copperlist con la stessa sfumatura, ma un poco "sfasata", e scambiarle continuamente.

Vediamo in pratica una *Dynamc Cop* in `Lezione11h1.s`. Avete notato la differenza? Si può spacciare per una sfumatura AGA! E dire che le copper dinamiche sono state usate in pochi giochi, nonostante non siano poi così difficili da fare. Tra i giochi che hanno copper dinamiche devo ricordare AGONY, e il recente SHADOW FIGHTER, il picchiaduro italiano dei NAPS TEAM.

Ora vediamo un'altra applicazione della copper2. Anziché scambiare un paio di copperlist, possiamo ciclarne qualche decina, per cui possiamo dire che si può "precalcolare" un effetto copper calcolando 1 copperlist per ogni "fase" dell'effetto: dato che poi l'effetto sarà ciclico, basterà puntare ogni volta alla copperlist "dopo". In questo modo otteniamo l'effetto copper, ma in termini di tempo risparmiamo TOTALMENTE quello che sarebbe stato usato dalle routines 68000! Per cui si può dire che l'effetto copper in questione è "GRATIS", e possiamo farci girare sopra una routine che mangia tutto il resto del tempo.

Vediamo una routine "normale", in `Lezione11h2.s`, e la versione che precalcola i fotogrammi "copper", ossia `Lezione11h3.s`. L'unico inconveniente della versione "turbizzata/precalcolata" è che serve della memoria aggiuntiva per memorizzare tutte le copperlist-frames.

Visto che volete superare l'esame di copper livello 2, dovete sapere anche che si può "mascherare" la coordinata Y dei `WAIT`. In pratica, un `WAIT` con la Y mascherata è così:

```
1       dc.w    $0007,$80FE     ; Wait ad Y "mascherata"
```

E significa: non controllare la linea Y, ma aspetta la posizione $07 X della linea attuale. è un `WAIT` "handicappato", che non sa leggere la posizione Y. In realtà non sa leggere i 7 bit bassi della posizione Y, per cui funziona dopo la linea $80. Ma allora, a cosa ci serve un wait mascherato che non controlla la posizione Y prima della posizione Y $80? Se avessimo da muovere una mitica barretta, di quelle della lezione 3, dovremmo cambiare tutti i wait che la compongono. Se invece mettiamo un wait normale all'inizio, e sotto tutti wait mascherati, basterà cambiare il primo wait e gli altri "seguiranno". Il risparmio di istruzioni 680x0 è evidente: con un solo `ADD/SUB` si sposta un'intera barretta.

Vediamo un'implementazione in `Lezione11h4.s` (mitica barretta di Lezione3!)

Da notare che funziona anche sotto la linea verticale $FF, dato che riparte la numerazione da $00. Ora che lo sapete, se vi capita di spostare qualche cosa in quella zona potete sfruttare questo trucchetto.

Ora si potrebbe parlare dell'istruzione `SKIP`, ma dato che non la ho mai vista usare da nessuno, e io stesso non vedo a cosa possa servire (si possono fare tranquillamente tutte le cose usando la copper2 per i salti...), tralascio questo argomento, spero crediate nella totale inutilità di tale comando.

Ora, per finire l'argomento *solo Copper senza bitplanes*, vi propongo 6 listati che riassumono i più comuni effetti di questo tipo.

`Lezione11i1.s` è uno scroll di colori a tutto schermo. `Lezione11i2.s` è una pseudo parallasse a 3 livelli di barre. Può servire come sfondo per un gioco platform durante una "salita", ad esempio. `Lezione11i3` è una fantasia in `COP` minore...`Lezione11i4.s` è una sfumatura pseudocasuale, che mixa i valori della posizione orizzontale del pennello elettronico (di solito valori diversi), per farne i colori della sfumatura copper. `Lezione11i5.s` è una copperlist che cicla i colori in un modo che sembri 3d.

11.9 Informazioni avanzate sul COPPER - Anche i BitPlanes abilitati

Avete visto che con i soli `MOVE&WAIT` del copper possiamo fare un bel po' di cose? Ma cosa succede se facciamo copperlist complicate con i bitplanes abilitati? Possiamo cambiare il `BPLMOD`

ad ogni linea per allungare le figure, o usare il bplcon1 ($dff102) per ondeggiarle, o addirittura cambiare ad ogni linea i puntatori ai bitplanes!!!

Nei listati seguenti, tra l'altro, è usato un sistema particolare per calcolare il DIWSTART/DIWSTOP, e DDFSTART/DDFSTOP, ossia tramite alcuni EQUATE, che sono usati per calcolare i valori grazie agli operatori di shift "«" e "»", nonché "&" (and), e i comuni "*", "/", "+", "-". Se volete fare uno schermo in 320*256 normale, si fa prima a mettere i valori normali, o cambiarli a mano. Se invece voleste fare uno schermo di grandezza particolare, ad esempio 256*256, può risparmiare tempo.

```
scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 1     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)
```

Poi, in copperlist metteremo:

```
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop
        dc.w    $100,BPLC0      ; BplCon0
```

Comunque, non è "infallibile", se volete fare schermi di grandezze strane potrebbe non funzionare, e sarà meglio andare "a mano". Si può anche usare per calcolare il valore, controllandolo dopo l'assemblaggio con `? DIWS` o `? xxxx`, poi scrivere il valore a mano. Ecco i listati di questa sezione:

`Lezione11l1.s` – Cambia ad ogni linea sia il `COLOR0` che il `BPLCON1` ($dff102), causando l'ondulazione dei bitplanes.

`Lezione11l2.s` – Si cambiano per ogni linea ben 3 colori su 4 (2 bitplanes).

`Lezione11l3a.s`, `Lezione11l3b.s` **e** `Lezione113c.s` – Sono 3 passaggi per arrivare a fare l'effetto di ondeggiamento del logo *AMIGA ET* della piccola demo presente nel disco 1. A forza di pezzi la abbiamo descritta tutta! La figura ondeggia grazie ai moduli negativi alternati a quelli azzerati.

`Lezione11l4.s` – Questo è un altro modo per ondeggiare: sono ridefiniti i bplpointers ad ogni linea!

`Lezione11l5.s` – Se aveste una piccola immagine larga 40*29 pixel, e voleste riempirci lo schermo intero cosa potreste fare? Provare a fare uno zoom di 8 volte, trasformandola in 320*232. è quello che fa questo listato, usando per l'allungamento verticale i modulo, per quello orizzontale una routine che testa ogni bit e lo "trasforma" in un byte (8 bit).

`Lezione11l5b.s` – È una versione ottimizzata del listato precedente, che usa una tabella contenente le 256 possibili combinazioni di un byte "espanso" a 8 byte. Richiede meno della metà di tempo per eseguire! Imparate a farvi questo tipo di ottimizzazioni: routines che sembrerebbero impossibili da rendere più veloci, alle volte possono subire una turbizzazione così!

11.10 Come fare uno schermo in interlace (lungo 512 linee)

Il modo interlacciato permette di visualizzare il doppio di dati video. Questo è possibile raddoppiando il numero di linee visualizzate. Normalmente sono possibili 256 linee verticali, mentre con l'interlace si può arrivare a 512, sia in lowres che in hires. Ci sono però delle particolarità, infatti non basta puntare i bitplanes e settare il bit di interlace (il bit 2 del `BPLCON0`). Per quanto riguarda la figura RAW, basta convertire un normale disegno interlacciato con l'iffconverter e salvarlo, ossia una figura in 320x512 o in 640x512. Anche un brush più piccolo va bene, ma considerate sempre che l'immagine non deve risultare "schiacciata" per la doppia risoluzione verticale. Come è noto l'interlacciato "sfarfalla", "traballa". Questo è un male, ma anche un bene, infatti su normali TV o monitor non sarebbe possibile una risoluzione verticale superiore a 256 linee, occorrerebbe un monitor "VGA", ossia multisync o multiscan. Il "trucchetto" è fatto visualizzando una volta solo le 256 linee dispari, e l'altra le altre 256 linee pari. Lo scambio avviene ogni frame per cui inganna decentemente l'occhio, a parte lo sfarfallio, (che, se sono scelti bene i colori, diminuisce molto). Questo scambio però non è del tutto "automatico", occorre fare qualcosina noi "a mano".

```
+----------------------------------+----------------------------------+
|    QUADRO 1 (linee dispari)      |        QUADRO 2 (linee pari)     |
+----------------------------------+----------------------------------+
| LINEA 1: ---> xxxxxxxxxxxx       |                                  |
|                                  |      LINEA 2: ---> xxxxxxxxxxxx  |
| LINEA 3: ---> xxxxxxxxxxxx       |                                  |
|                                  |      LINEA 4: ---> xxxxxxxxxxxx  |
| LINEA 5: ---> xxxxxxxxxxxx       |                                  |
|                                  |      LINEA 6: ---> xxxxxxxxxxxx  |
| LINEA 7: ---> xxxxxxxxxxxx       |                                  |
|                                  |      LINEA 8: ---> xxxxxxxxxxxx  |
| LINEA 9: ---> xxxxxxxxxxxx       |                                  |
|                                  |      LINEA 10: --> xxxxxxxxxxxx  |
|          [...]                   |                                  |
|                                  |                [...]             |
| LINEA 311: -> xxxxxxxxxxxx       |                                  |
|                                  |      LINEA 312: -> xxxxxxxxxxxx  |
| LINEA 313: -> xxxxxxxxxxxx       |                                  |
+----------------------------------+----------------------------------+
```

Per il modo interlacciato occorre ridefinire il modulo mettendolo a 40 se siamo in lowres, o a 80 se siamo in hires. In pratica occorre mettere nel modulo la lunghezza di una linea, per saltarla: essendo il modulo un valore che viene aggiunto alla fine di ogni linea video, se saltiamo la lunghezza di una intera linea ecco cosa succede: viene letta e visualizzata la prima linea, alla fine viene saltata la seconda, e sotto viene visualizzata la terza; alla fine di questa viene saltata una linea e viene visualizzata la quinta, eccetera. In pratica abbiamo fatto in modo che siano

visualizzate solo le linee dispari. Lo schermo per motivi hardware non può visualizzare più di 256 linee, ma nessuno ci impone quali 256, nè da quando farle iniziare a visualizzare. Se una volta saltiamo le linee pari, e quella dopo saltiamo le linee dispari, possiamo visualizzare una schermata di 512 linee in una da 256 "alternata"!

Ricapitoliamo: abbiamo una pic, per esempio, in 640x512 interlacciata, che abbiamo convertito in RAW e vogliamo visualizzare. Abbiamo puntato la pic opportunamente e messo il modulo a -80, nonché settato il bit di interlace (oltre a quello di hires) nel bplcon0 ($dff100). Cosa otteniamo? **La figura come fosse in lowres! Alta 256 linee, come se l'avessimo abbassata di risoluzione!**

Allora, qua è il momento di fare quella piccola parte "a mano" per permettere l'interlacciamento. Esiste un apposito bit, che va controllato, che ci indica se visualizzare le linee dispari o quelle pari a ogni fotogramma. Questo è il bit 15 del VPOSR ($dff004), detto LOF, o Long Frame, che indica se siamo nel "frame lungo" o no. Ecco un esempio di routine:

```
1  LACEINT:
2          MOVE.L  #BITPLANE,D0     ; Indirizzo bitplane
3          btst.b  #15-8,$dff004    ; VPOSR LOF bit?
4          Beq.S   Faidispari       ; Se si, tocca alle linee dispari
5          ADD.L   #80,D0           ; Oppure aggiungi la lunghezza di una linea,
6                                   ; facendo partire la visualizzazione dalla
7                                   ; seconda: visualizzate linee pari!
8  FaiDispari:
9          LEA     BPLPOINTERS,A1   ; PLANE POINTERS IN COPLIST
10         MOVE.W  D0,6(A1)         ; Punta la figura
11         SWAP    D0
12         MOVE.W  D0,2(A1)
13         RTS
```

Come vedete, se il bit LOF è azzerato si parte a visualizzare dalla prima linea, quindi per effetto del modulo saranno visualizzate le linee 1,3,5,7... eccetera, ossia quelle dispari. Altrimenti, si salta una linea, facendo partire la visualizzazione dalla seconda, quindi 2,4,6,8... eccetera: pari!

C'è chi fa 2 copperlist, una che punta in un modo e una che punta nell'altro, e poi a seconda del bit LOF punta una o l'altra ogni fotogramma. Credo però sia più "furbo" puntare solo i bitplanes... comunque potete fare come volete, basta aver capito il metodo.

`Lezione11l6.s` – è un esempio in 640x512 ad 1 bitplane

`Lezione11l6b.s` – è un esempio in 320x512 a 4 bitplanes

Per terminare questo corso di laurea in "copper", livello 2, non ci rimane che fare un esempio più complesso anche per gli sprite. Vi ricordate come li "riutilizzammo" nella lezione7 per fare le stelline? Ebbene, cosa succede se riutilizziamo gli sprite ogni 2 linee?

`Lezione11l7.s` – Per vedere un mega riutilizzo degli sprite (128 volte ognuno)

11.11 I due chip 8520, detti CIAA e CIAB

Se smontate l'Amiga troverete, oltre ad Agnus, Paula, Denise, il 680x0 ecc., anche un paio di chippetti 8520, detti CIA. Questi chip hanno 16 pin di Input/Output ognuno, un registro di shift seriale, tre timer, un pin di solo output e uno di solo input. Di conseguenza entrambi hanno 16 registri che si possono raggiungere accedendo ai loro relativi indirizzi:

```
Mappa di indirizzi del CIAA
--------------------------------------------------------------------------
 Byte    Register                 Data bits
Address    Name     7     6     5     4     3     2     1     0
```

```
------------------------------------------------------------------------
$BFE001    pra     /FIR1 /FIR0  /RDY /TK0  /WPRO /CHNG /LED  OVL
$BFE101    prb     Porta parallela
$BFE201    ddra    Direzione per port A (BFE001);1=output (normalmente $03)
$BFE301    ddrb    Direzione per port B (BFE101);1=output (può essere in/out)
$BFE401    talo    CIAA timer A byte basso (.715909 Mhz NTSC; .709379 Mhz PAL)
$BFE501    tahi    CIAA timer A byte alto
$BFE601    tblo    CIAA timer B byte basso (.715909 Mhz NTSC; .709379 Mhz PAL)
$BFE701    tbhi    CIAA timer B byte alto
$BFE801    todlo   Timer a 50/60 Hz - bits 7-0 (VSync or line tick)
$BFE901    todmid  Timer a 50/60 Hz - bits 15-8
$BFEA01    todhi   Timer a 50/60 Hz - bits 23-16
$BFEB01            Non usato
$BFEC01    sdr     CIAA serial data register (connesso alla tastiera)
$BFED01    icr     CIAA interrupt control register
$BFEE01    cra     CIAA control register A
$BFEF01    crb     CIAA control register B

Nota:  il CIAA può generare un interrupt INT2, ossia liv.2, $68.


Mappa di indirizzi del CIAB
------------------------------------------------------------------------
 Byte     Register                  Data bits
Address     Name     7     6     5     4     3     2     1     0
------------------------------------------------------------------------
$BFD000    pra     /DTR  /RTS  /CD   /CTS  /DSR   SEL   POUT  BUSY
$BFD100    prb     /MTR  /SEL3 /SEL2 /SEL1 /SEL0 /SIDE  DIR  /STEP
$BFD200    ddra    Direction for Port A (BFD000);1 = output (normalm. a $FF)
$BFD300    ddrb    Direction for Port B (BFD100);1 = output (normalm. a $FF)
$BFD400    talo    CIAB timer A byte basso (.715909 Mhz NTSC; .709379 Mhz PAL)
$BFD500    tahi    CIAB timer A byte alto
$BFD600    tblo    CIAB timer B byte basso (.715909 Mhz NTSC; .709379 Mhz PAL)
$BFD700    tbhi    CIAB timer B byte alto
$BFD800    todlo   Timer di sync orizzontale -  bits 7-0
$BFD900    todmid  Timer di sync orizzontale -  bits 15-8
$BFDA00    todhi   Timer di sync orizzontale -  bits 23-16
$BFDB00            Non usato
$BFDC00    sdr     CIAB serial data register (non usato)
$BFDD00    icr     CIAB interrupt control register
$BFDE00    cra     CIAB Control register A
$BFDF00    crb     CIAB Control register B

Nota:  CIAB può generare un INT6, ossia liv 6: $78.
```

Da questa “mappa” si può vedere come i compiti dei 2 CIA vadano dalla lettura della tastiera, alla gestione della porta seriale, (per scambio dati tra 2 computer o tra computer e modem), alla gestione della porta parallela (per la stampante, ad esempio), al controllo delle testine del disk drive, inoltre ha degli "orologi" che possono contare microsecondi o ore.

In realtà però non ci interesseranno tutte queste caratteristiche, per vari motivi. Innanzitutto, la parte riguardante l’hardware dei disk drive può essere omessa, dato che ogni buon gioco/demo che si rispetti deve essere installabile su Hard Disk (o CD-ROM!), come ad esempio Brian The Lion. Per quanto riguarda la gestione della porta parallela e della porta seriale, gli utilizzi potrebbero essere quello di stampare qualche istruzione per il gioco o qualche frase detta da un personaggio, oppure per la seriale la possibilità di giocare in due con i computer in rete, ossia connessi con un cavo. Però occorre dire che la gestione della stampante è bene farla fare al sistema operativo, usando il `parallel.device`. Lo stesso dicasi della porta seriale: il `serial.device` è certamente più sicuro di routines scritte via hardware, specialmente per gli Amiga futuri o

per le schede multiseriali. Per quanto riguarda i timers, dato che il sistema operativo ne utilizza diversi per le proprie mansioni, vedremo se e quali utilizzare. Ma allora, ci interessa quasi esclusivamente la lettura da tastiera? Ebbene si, infatti se sbirciaste nel codice di un videogioco, notereste che vengono ridefiniti solo gli interrupt $6c (COPER/VERTB/VBLANK) e $68 (INT2 del ciaa di tastiera): il livello 3 ($6c) serve per mettere la musica o altre routines in sincronia col pennello elettronico, mentre il livello 2 ($68) per leggere la tastiera. Naturalmente con la lettura del tasto sinistro del mouse o di altre cose si accede a registri cia, ma si tratta di semplici controlli o settaggi di BIT, non occorre fare lunghe dissertazioni su `btst.b #6,$bfe001`. Nelle demo, addirittura, è facile che non sia ridefinito alcun interrupt, oppure che sia usato solo il $6c per metterci la musica.

Quindi, cominciamo la spiegazione del CIAA dalla gestione della tastiera, in cui sono coinvolti i registri $bfec01 (SDR), $bfed01 (ICR), $bfee01 (CRA) e l'interrupt di livello 2 ($68). Prima vediamo i 3 registri separatamente, poi facciamo degli esempi sul loro corretto utilizzo. Premetto che quando un tasto viene premuto o rilasciato viene inviato dalla tastiera un codice a 8 bit attraverso $bfec01, e viene generato un interrupt di livello 2 ($68) nel quale occorre "dire" alla tastiera che si è ricevuto il codice di quel tasto. Da notare che tale codice NON è il codice ascii del carattere premuto, ma un codice con la posizione del bottone premuto nella tastiera.

```
*****************************************************************************
;* BFEC01    sdr     CIAA sdr (serial data register - connesso alla tastiera)
*****************************************************************************
```

è un registro di shift a 8 bit sincrono, connesso alla tastiera. Esso può funzionare in 2 modi: INPUT o OUTPUT, e la selezione tra questi due modi si può fare agendo sul bit 6 di $bfee01 (CRA). Nel modo INPUT i dati ricevuti dalla tastiera sono inseriti nel registro, un bit alla volta e, quando sono "arrivati" tutti gli 8 bit che formano il carattere premuto, si genera un'interrupt INT2 ($68), dal quale occorre vedere di quale tasto si tratta e annotarlo in qualche variabile. In questo caso si legge il byte corrispondente al codice del carattere: Es:

```
1 move.b $bfec01,d0
```

Nel modo OUTPUT invece si scrive sul registro, ad esempio `clr.b $bfec01`.

```
*****************************************************************************
;* BFED01    icr     CIAA interrupt control register
*****************************************************************************
```

Questo registro controlla gli interrupt generabili dal CIAA. I Cia infatti generano degli interrupt in varie occasioni, per esempio quando è finito un conto alla rovescia di un timer, o quando la porta seriale ha finito un trasferimento. A noi in particolare interessa l'interrupt INT2, di livello2, ossia il vettore di offset $68, che viene generato quando è premuto un tasto. Il funzionamento degli icr ($bfed01 per CIAA e $bfdd00 per CIAB) è molto particolare, infatti consistono in una "maschera" a sola scrittura di un registro dati a sola lettura. Ma cosa significa questo? Innanzitutto che è molto facile sbagliarsi e far impazzire gli interrupt del CIA, il che è poco augurabile. Ogni interrupt risulta abilitato se il corrispondente bit della maschera è settato ad 1, infatti ogni interrupt CIAA, come farebbe con un INTREQ ($dff09c), setta il suo bit di richiesta in questo registro. A questo punto, se tale interrupt è abilitato, si setta il bit 7 (IR), che è una specie di SET/CLR bit, come in DMACON, ossia quando tale bit è azzerato gli altri 6 bit settati sono azzerati, quando il bit 7 è settato, invece, gli altri bit settati sono settati, mentre quelli a zero non vengono modificati. La cosa che può confondere è che quando si legge il registro il suo contenuto viene azzerato, sia che si faccia un `tst.b $bfed01` che qualsiasi azione di lettura; azzerando il registro si elimina anche la richiesta di interrupt, in modo analogo all'azzeramento

dei bit di INTREQ ($dff09c). Ora ci interessa solo la sua funzione per l'interrupt di tastiera, per cui vediamo in breve i suoi bit in modo lettura, con commenti solo dove interessa:

```
CIAA ICR ($bfed01)

BIT     NOME    DESCRIZIONE

07      IR      Bit che indica, se settato, che c'è un interrupt in corso
06      0
05      0
04      FLG
03      SP      Se settato, siamo in un interrupt generato dalla tastiera
02      ALRM
01      TB
00      TA
```

Ricordarsi che se si legge il registro questo si azzera, per cui se volete sapere quali bit fossero settati, dovete copiarlo in un registro Dx e fare dei controlli su tale registro: rileggendo $bfed01 i bit sono azzerati.

```
*****************************************************************************
;* BFEE01    cra     CIAA cra (control register A)
*****************************************************************************
```

Questo registro è detto "di controllo" proprio perché i suoi bit controllano la funzione di altri registri. Ecco una sua "mappa", con i commenti solo ai bit che ci interessano per la lettura da tastiera:

```
CIA Control Register A

  BIT  NOME     FUNZIONE
  ---  ----     --------
   0  START     Timer A
   1  PBON      Timer A
   2  OUTMODE   Timer A
   3  RUNMODE   Timer A
   4  LOAD      Timer A
   5  INMODE    Timer A
   6  SPMODE    Se è a 1 = registro ($bfec01) output (per scriverci)
                Se è a 0 = registro ($bfec01) input (per leggerlo)
   7   Non usato
```

Come si vede, l'unico bit che ci interessa è il 6, che "decide" la funzione del $bfec01, ossia se la sua direzione sia "verso la tastiera" (output), per cui possiamo scriverci, o "dalla tastiera verso l'Amiga" (input), per cui possiamo leggere il carattere relativo al tasto che è stato premuto. Per cambiare modalità basta fare questo:

```
1       bset.b  #6,$bfee01      ; CIAA cra - sp ($bfec01) output
2       ....
3       bclr.b  #6,$bfee01      ; CIAA cra - sp (bfec01) input
```

Oppure, se vi pare più elegante, potete usare AND e OR per farlo:

```
1       or.b    #$40,$bfee01    ; SP OUTPUT (%0100000, settiamo il bit 6!)
2       ...
3       and.b   #$bf,$bfee01"   ; SP INPUT  (%10111111, azzeriamo il bit 6!)
```

Potreste anche muovere "0000" in un registro, moltiplicarlo per 5, dividerlo per 5, aggiungere 20, sottrarre 10, aggiungere 1, sottrarre 11, settare o azzerare il bit 6, e fare un and o un or con il *$bfee01*, l'assembler permette di usare infinite strade per fare la stessa cosa. Ma il BSET/CLR

può bastare! Comunque occorre precisare che tra modo input e modo output occorre attendere una novantina di microsecondi, dato che l'hardware del CIAA e del chip della tastiera non può temporizzarsi da solo in modo input. Gli 8 bit del carattere corrispondente al tasto premuto sono trasferiti serialmente un bit alla volta dal chip della tastiera al CIAA. Quando tutti e 8 i bit sono stati trasferiti, **dobbiamo abbassare la linea** KDAT **per almeno una novantina di microsecondi (o 3/4 linee raster) per confermare alla tastiera che abbiamo ricevuto i dati**. Il "filo" KDAT si controlla dal bit SP/SPMODE, e in pratica dobbiamo fare questo:

```
1  ------------------------------------------------------------------------
2          move.b  $bfec01,d0      ; CIAA sdr - Leggiamo il carattere attuale
3          bset.b  #6,$bfee01      ; CIAA cra - sp ($bfec01) output, in modo da
4                                  ; abbassare la linea KDAT per confermare che
5                                  ; abbiamo ricevuto il carattere.
6
7          st.b    $bfec01         ; $FF in $bfec01 - uè! ho ricevuto il dato!
8
9  ; Qua dobbiamo mettere una routine che aspetti 90 millisecondi perchè la
10 ; linea KDAT deve stare bassa abbastanza tempo per essere "capita" da tutti
11 ; i tipi di tastiere. Si possono, per esempio, aspettare 3 o 4 linee raster.
12
13         bclr.b  #6,$bfee01      ; CIAA cra - sp (bfec01) input nuovamente.
14 ------------------------------------------------------------------------
```

Leggendo la tastiera via hardware, occorre stare molto attenti alla temporizzazione che attende 90 millisecondi, per 2 motivi:

1. La routine di temporizzazione deve aspettare lo stesso tempo su tutti i processori, dal 68000 al 68060. Per questo potete usare il pennello elettronico, o anche un timer del CIA, ma **mai** fare un semplice loop DBRA eseguito molte volte, o una serie di NOP, perché su 68020+ a causa della cache sarà eseguito in un battibaleno.

2. Una volta che la nostra routine "aspetta" bene su tutti i 680x0, occorre anche considerare il fatto che non tutte le tastiere sono uguali! Per esempio per una tastiera potrebbero bastare 2 linee raster mentre per un'altra ne potrebbero occorrere 4! Infatti le tastiere contengono un chip che le controlla, e questo può essere diverso nei diversi modelli di Amiga.

Per esempio nell'A1200 la tastiera è "economica", infatti si differenzia dalle tastiere Amiga normali (Mitsumi in genere) per il fatto che non si può registrare più di una pressione alla volta... se si tiene premuto un tasto e nel contempo se ne preme un altro, rilasciando il primo non compare il secondo. La routine di wait che deve aspettare tra: `or.b #$40` o `bset.b #6, $bfee01` e `and.b #$bf` o `bclr.b #6",$bfee01` determina se il vostro programma leggerà la tastiera correttamente o si inchioderà alla pressione di un tasto su alcuni computer.

A tal proposito vediamo in che modo aspettare correttamente, usando il VBLANK:

```
1  ; Se non volete "sporcare" registri indirizzi:
2
3  ------------------------------------------------------------------------
4          moveq   #4-1,d0         ; we wait 4 rasterlines (3+random...!)
5  waitlines:
6          move.b  $DFF006,d1
7  stepline:
8          cmp.b   $DFF006,d1
9          beq.s   stepline
10         dbra    d0,waitlines
11 ------------------------------------------------------------------------
12
13 ; Se invece volete "sporcare" anche un registro indirizzi:
14
15 ------------------------------------------------------------------------
16         lea     $dff006,a0      ; VHPOSR
17         moveq   #4-1,d0 ; Numero di linee da aspettare = 4 (in pratica 3 più
18                         ; la frazione in cui siamo nel momento di inizio)
```

```
19 waitlines:
20         move.b  (a0),d1 ; $dff006 - linea verticale attuale in d1
21 stepline:
22         cmp.b   (a0),d1         ; siamo sempre alla stessa linea?
23         beq.s   stepline        ; se si aspetta
24         dbra    d0,waitlines    ; linea "aspettata", aspetta d0-1 linee
25 ------------------------------------------------------------------------
```

Caratteristiche del codice carattere trasmesso in *$bfec01*

Abbiamo già detto che il codice trasmesso non è un codice ascii, ma una informazione sul tasto che è stato premuto. Questo anche perché a seconda delle diverse tastiere, in lingua inglese, italiana o altro, molti tasti hanno una lettera diversa stampata sopra. Ma se dico: il terzo tasto della seconda fila, non ci si può sbagliare. Comunque gli 8 bit (1 byte) che prendiamo dal $bfec01 contengono 7 bit relativi all'identificatore del tasto, più un bit che stabilisce se il tasto è stato premuto o rilasciato. Infatti, l'identificatore del tasto viene inviato sia quando si preme che quando si rilascia, con la differenza che il bit più alto, l'ottavo, una volta è azzerato (rilasciato) o settato (premuto).

Come non bastasse, tutti i codici trasmessi sono ruotati di un bit a sinistra prima della loro trasmissione. L'ordine della trasmissione quindi è 6-5-4-3-2-1-0-7. Comunque basta usare l'istruzione "ROR.B #1,xxx" per riportare l'ordine 7-6-5-4-3-2-1-0. La trasmissione di un bit impiega 60 microsecondi, dunque l'intero byte che costituisce il carattere viene trasferito in 480 microsecondi, per cui si possono trasferire 17000 bit al secondo. Ma che ci importa? Nulla! Vediamo invece come si fa a riconoscere se il tasto premuto è la A, la B o qualche altro. Nell'hardware manual c'è una lista con un codice per tasto, dove la particolarità è che al codice $01 corrisponde il tasto "1". Per ottenere questi codici occorre eseguire un `NOT` del byte, oltre che fare una rotazione dei bit con un `ROR` per riportare l'ordine 76543210. In pratica, quello che dobbiamo fare è:

```
1         move.b  $bfec01,d0      ; CIAA sdr (serial data register - connesso
2                                 ; alla tastiera - contiene il byte inviato dal
3                                 ; chip della tastiera) LEGGIAMO IL CHAR!
4         NOT.B   D0              ; aggiustiamo il valore invertendo i bit
5         ROR.B   #1,D0           ; e riportando la sequenza a 76543210.
```

Ora in d0 abbiamo il byte con la sequenza di bit 76543210 anziché 65432107, e in più tutti i bit sono invertiti, in modo da far partire il "conto" dal primo tasto in alto a sinistra (non ESC, ma quello accanto all'1). Ecco la sequenza dei codici con il relativo carattere (normale e shiftato, ma considerate che qua la tastiera descritta è quella USA). (TASTI PREMUTI)

```
cod.    $00     ;` - ~
cod.    $01     ;1 - !
cod.    $02     ;2 - @
cod.    $03     ;3 - #
cod.    $04     ;4 - $
cod.    $05     ;5 - %
cod.    $06     ;6 - ^
cod.    $07     ;7 - &
cod.    $08     ;8 - *
cod.    $09     ;9 - (
cod.    $0A     ;0 - )
cod.    $0B     ;- - _
cod.    $0C     ;= - +
cod.    $0D     ;\ - |
cod.    $0e     ;  << vuoto
cod.    $0F     ;0  tastierino numerico
cod.    $10     ;q - Q
cod.    $11     ;w - W
cod.    $12     ;e - E
```

```
cod.    $13     ;r - R
cod.    $14     ;t - T
cod.    $15     ;y - Y
cod.    $16     ;u - U
cod.    $17     ;i - I
cod.    $18     ;o - O
cod.    $19     ;p - P
cod.    $1A     ;[ - {
cod.    $1B     ;] - }
cod.    $1c     ; << non usato
cod.    $1D     ;1  tastierino numerico
cod.    $1E     ;2  tastierino numerico
cod.    $1F     ;3  tastierino numerico
cod.    $20     ;a - A
cod.    $21     ;s - S
cod.    $22     ;d - D
cod.    $23     ;f - F
cod.    $24     ;g - G
cod.    $25     ;h - H
cod.    $26     ;j - J
cod.    $27     ;k - K
cod.    $28     ;l - L
cod.    $29     ;; - :
cod.    $2A     ;' - "
cod.    $2B     ;(solo in tast. internazionali) - vicino al return
cod.    $2c     ; << non usato
cod.    $2D     ;4  tastierino numerico
cod.    $2E     ;5  tastierino numerico
cod.    $2F     ;6  tastierino numerico
cod.    $30     ;< (shift sin. solo in tastiere internazionali)
cod.    $31     ;z - Z
cod.    $32     ;x - X
cod.    $33     ;c - C
cod.    $34     ;v - V
cod.    $35     ;b - B
cod.    $36     ;n - N
cod.    $37     ;m - M
cod.    $38     ;, - <
cod.    $39     ;. - >
cod.    $3A     ;/ - ?
cod.    $3b     ; << non utilizzato
cod.    $3C     ;.  tastierino numerico
cod.    $3D     ;7  tastierino numerico
cod.    $3E     ;8  tastierino numerico
cod.    $3F     ;9  tastierino numerico
cod.    $40     ;space
cod.    $41     ;back space <-
cod.    $42     ;tab ->|
cod.    $43     ;return tastierino numerico (enter)
cod.    $44     ;return <-'
cod.    $45     ;esc
cod.    $46     ;del
cod.    $47     ; << non usato
cod.    $48     ; << non usato
cod.    $49     ; << non usato
cod.    $4A     ;-  tastierino numerico
cod.    $4b     ; <<
cod.    $4C     ;cursore alto  ^
cod.    $4D     ;cursore basso v
cod.    $4E     ;cursore destra   »
cod.    $4F     ;cursore sinistra «
cod.    $50     ;f1
```

```
        cod.    $51     ;f2
        cod.    $52     ;f3
        cod.    $53     ;f4
        cod.    $54     ;f5
        cod.    $55     ;f6
        cod.    $56     ;f7
        cod.    $57     ;f8
        cod.    $58     ;f9
        cod.    $59     ;f10
        cod.    $5A     ;(  tastierino numerico
        cod.    $5B     ;)  tastierino numerico
        cod.    $5C     ;/  tastierino numerico
        cod.    $5D     ;*  tastierino numerico
        cod.    $5E     ;+  tastierino numerico
        cod.    $5F     ;help
        cod.    $60     ;lshift (sinistro)
        cod.    $61     ;rshift (destro)
        cod.    $62     ;caps lock
        cod.    $63     ;ctrl
        cod.    $64     ;lalt (sinistro)
        cod.    $65     ;ralt (destro)
        cod.    $66     ;lamiga (sinistro)
        cod.    $67     ;ramiga (destro)
```

Come vedete, la sequenza rispetta grossomodo l'ordine dei tasti, partendo dalla fila con 1,2,3,4,5... poi la fila sotto con q,w,e,r,t,y... eccetera. Questi codici si riferiscono ai tasti **premuti**, dove il bit 7, quello che determina se un tasto è stato premuto o rilasciato, è a zero. Infatti abbiamo eseguito un `NOT` sul byte, che ha invertito pure il bit 8: se un tasto è **premuto**, ora il bit 8=0, se un tasto è rilasciato, il bit 8=1. Se il tasto è rilasciato, il bit 7 (l'ottavo) va considerato come settato, per cui la tabella di prima diventerebbe:

```
        cod.    $80     ;` - ~
        cod.    $81     ;1 - !
        cod.    $82     ;2 - @
        ...
```

Attenzione al fatto che gli Amiga600 non hanno il tastierino numerico, per cui se usate i tasti del tastierino su tali computer non c'è modo di premerli! Vi consiglio quindi di evitare i tasti del tastierino. Alla luce di queste considerazioni, possiamo vedere come sarà l'interrupt di livello 2 ($68), che ci permetterà di salvare nella variabile "ActualKey" il codice dei tasti premuti:

11.12 Routine di interrupt $68 (livello 2) - gestione tastiera

```
1  ;03     PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2
2
3  MioInt68KeyB:   ; $68
4          movem.l d0-d1/a0,-(sp)  ; salva i registri usati nello stack
5          lea     $dff000,a0      ; reg. custom per offset
6
7          MOVE.B  $BFED01,D0      ; Ciaa icr - in d0 (leggendo l'icr causiamo
8                                  ; anche il suo azzeramento, per cui l'int è
9                                  ; "disdetto" come in intreq).
10         BTST.l  #7,D0   ; bit IR, (interrupt cia autorizzato), azzerato?
11         BEQ.s   NonKey  ; se si, esci
12         BTST.l  #3,D0   ; bit SP, (interrupt della tastiera), azzerato?
13         BEQ.s   NonKey  ; se si, esci
14
15         MOVE.W  $1C(A0),D0      ; INTENAR in d0
16         BTST.l  #14,D0          ; Bit Master di abilitazione azzerato?
17         BEQ.s   NonKey          ; Se si, interrupt non attivi!
```

```
18          AND.W   $1E(A0),D0      ; INREQR - in d1 rimangono settati solo i bit
19                                  ; che sono settati sia in INTENA che in INTREQ
20                                  ; in modo da essere sicuri che l'interrupt
21                                  ; avvenuto fosse abilitato.
22          btst.l  #3,d0           ; INTREQR - PORTS?
23          beq.w   NonKey          ; Se no, allora esci!
24
25  ; Dopo i controlli, se siamo qua significa che dobbiamo prendere il carattere!
26
27          moveq   #0,d0
28          move.b  $bfec01,d0      ; CIAA sdr (serial data register - connesso
29                                  ; alla tastiera - contiene il byte inviato dal
30                                  ; chip della tastiera) LEGGIAMO IL CHAR!
31
32  ; abbiamo il char in d0, lo "lavoriamo"...
33
34          NOT.B   D0              ; aggiustiamo il valore invertendo i bit
35          ROR.B   #1,D0           ; e riportando la sequenza a 76543210.
36          move.b  d0,ActualKey    ; salviamo il carattere
37
38  ; Ora dobbiamo comunicare alla tastiera che abbiamo preso il dato!
39
40          bset.b  #6,$bfee01      ; CIAA cra - sp ($bfec01) output, in modo da
41                                  ; abbassare la linea KDAT per confermare che
42                                  ; abbiamo ricevuto il carattere.
43
44          st.b    $bfec01         ; $FF in $bfec01 - uè! ho ricevuto il dato!
45
46  ; Qua dobbiamo mettere una routine che aspetti 90 microsecondi perchè la
47  ; linea KDAT deve stare bassa abbastanza tempo per essere "capita" da tutti
48  ; i tipi di tastiere. Si possono, per esempio, aspettare 3 o 4 linee raster.
49
50          moveq   #4-1,d0 ; Numero di linee da aspettare = 4 (in pratica 3 più
51                          ; la frazione in cui siamo nel momento di inizio)
52  waitlines:
53          move.b  6(a0),d1        ; $dff006 - linea verticale attuale in d1
54  stepline:
55          cmp.b   6(a0),d1        ; siamo sempre alla stessa linea?
56          beq.s   stepline        ; se si aspetta
57          dbra    d0,waitlines    ; linea "aspettata", aspetta d0-1 linee
58
59  ; Ora che abbiamo atteso, possiamo riportare $bfec01 in modo input...
60
61          bclr.b  #6,$bfee01      ; CIAA cra - sp (bfec01) input nuovamente.
62
63  NonKey:         ; 3210
64          move.w  #%1000,$9c(a0)  ; INTREQ togli richiesta, int eseguito!
65          movem.l (sp)+,d0-d1/a0  ; ripristina i registri dallo stack
66          rte
67
68  ------------------------------------------------------------------------
```

Non dovreste aver notato niente di nuovo, è solo una "sintesi" delle cose che abbiamo già spiegato. Poi in definitiva sono solo poche linee e usiamo solo i registri d0 ed a0, non è mica una routine *complicata*!. L'unica cosa che dovete ricordarvi è di mettere questo interrupt al vettore $68+VBR, e di abilitarlo, settando il bit 3 di `INTENA` ($dff09a). Per esempio, se state usando un'interrupt di livello 3 ($6c) per suonare la musica, usando il solo `VERTB` (bit 5), potete scrivere:

```
1                   ; 5432109876543210
2           move.w  #%1100000000101000,$9a(a5)    ; INTENA - abilito solo VERTB
3                                                 ; del livello 3 e il livello2
```

O in altri termini `move.w #$c028,$dff09a`. Possiamo vedere il corretto uso di questo interrupt in `Lezione11m1.s`.

In questo listato il codice della tastiera viene immesso nel `COLOR0` per far "vedere" l'effettivo funzionamento della routine stessa. Per uscire naturalmente occorre premere un tasto: la barra di spazio.

Per comodità, in `Lezione11m2.s` è inclusa una routine rudimentale di conversione dei codici di tastiera in ASCII, che può servire se volete fare in modo che si possa stampare quello che viene scritto con la tastiera, per esempio se volete farvi una utility, o semplicemente scrivere il vostro nome nell'high score del vostro gioco.

11.13 I timers del CIAA e del CIAB

Questi timers in realtà sono usati pochissimo nei giochi, e nelle demo quasi mai, giusto in certe (complicate) routines che suonano la musica, che comunque basta includere e suonano per i fatti loro. Tra l'altro questi timers sono usati anche dal sistema operativo, per cui usandoli si può rischiare di far impazzire tutto all'uscita del nostro programma. Inoltre, aspettando le linee raster con il $dff006 si possono fare tutte le attese che servono, senza rischiare queste eventualità. Per questo, nella lezione sono presenti solo un paio di listati che usano i timer, come esempio. In listati di lezioni più avanzate avremo modo di trovare applicazioni di questi timer, e li vedremo caso per caso.

`Lezione11n1.s` – Uso del timer A del CIAA o CIAB

`Lezione11n1b.s` – Uso del timer B del CIAA o CIAB

`Lezione11n2.s` – Uso del timer TOD (Time of day)

Nell'usare i timer del CIA considerate che il sistema operativo usa questi per i seguenti scopi: (meglio usare il CIAB!)

```
   CIAA, timer A        Utilizzato per l'interfacciamento con la tastiera

!  CIAA, timer B        Utilizzato dall'exec per lo scambio dei task ecc.

   CIAA, TOD            timer a 50/60 Hz utilizzato dal Timer.device

   CIAB, timer A        Non utilizzato, disponibile per i programmi

   CIAB, timer B        Non utilizzato, disponibile per i programmi

   CIAB, TOD            Utilizzato dalla graphics.library per seguire le
                        posizioni del pennello elettronico.
```

Se dovete usare timer che servono anche al sistema operativo, fatelo solo se avete disabilitato multitasking e interrupt di sistema, se avete cioè preso il controllo completo del sistema. Mai il CIAA, timer B!

11.14 Il caricamento di files con la `dos.library`

Per concludere questa lezione piena di perfezionamenti e argomenti vari, non c'è miglior argomento del **caricamento dei dati**. Se volete programmare qualcosa di "grosso", dal punto di vista della mole di disegni, musiche e dati vari, non si può semplicemente includere tutto con l'incbin e salvare il mega eseguibile con "WO", perchè il file verrebbe troppo grande per poter essere caricato in memoria. Supponiamo, per esempio, di voler fare uno slideshow, ossia un programma che mostri una serie di figure una dopo l'altra, e che queste figure siano ben 30, lunghe 100Kb l'una. Vengono 3MB di figure. Non potendo fare una serie di 30 INCBIN per salvare un file di 3MB e passa, non ci resta che trovare un modo per "caricarne" una alla volta. Ma quale modo usare? Ce ne sono 2 principalmente:

Caricamento AutoBoot dalle tracce del dischetto ossia una modalità non compatibile col DOS, infatti potrete notare che molti dischi di giochi, se inseriti nel drive una volta caricato il workbench, non sono leggibili con comandi come "DIR", e risultano NON DOS, o ERRATI... insomma sembrano dischi marci! Se sottoposti a copia, tramite copiatori come XCOPY o DCOPY, alcuni di questi giochi non dos appaiono a tracce "ROSSE", ossia non riconoscibili nemmeno dal copiatore, mentre altri nonostante risultino illeggibili dal dos appaiono come "SANI", ossia a tracce verdi. Devo precisare che i giochi CRACKATI (sprotetti e distribuiti dai pirati) sono tutti del secondo tipo, ossia a tracce VERDI, infatti la sprotezione spesso sta nel trasformare le tracce incopiabili in tracce copiabili, ma spesso rimangono illeggibili dal dos. Le *TRACKMO* sono la gran parte delle demo, e sono a tracce "copiabili", ma non leggibili via dos. Una caratteristica è quella che occorre scrivere codice per indirizzi assoluti, non rilocabile, per cui solitamente viene usato solo il primo mega di CHIP RAM, o per a1200 i primi 2 mega, ed eventuali espansioni di FAST RAM non sono utilizzate, a parte quelli che usano COMPLESSI LOADER CON RILOCATORI che somigliano a mini sistemi operativi, che però spesso fanno cilecca sul 68040 per le eccessive "esuberanze" di programmazione. Questo sistema ha il "pregio" di essere leggermente più veloce, sui floppy disk, del dos normale, ma lo svantaggio di non poter installare su Hard-Disk il programma, nè poterlo convertire per CD32 eccetera.

Caricamento "legale" usando la `dos.library` ossia un modo molto simile a quello usato da un qualsiasi programma che usa il sistema operativo, compilato con un qualsiasi linguaggio come il C, l'AMOS eccetera. In realtà, mantenendo una nostra copperlist e operando sui registri hardware facciamo un sistema "ibrido", ossia usamo la `dos.library` in uno stato "particolare" con la nostra copperlist e i nostri interrupt. Una caratteristica dei programmi che usano questo sistema è che il sistema operativo deve essere lasciato "intatto" e il codice deve essere totalmente rilocabile, (potendo accedere all'eventuale FAST RAM). Questo sistema ha il pregio di poter essere usato su Hard Disk, CD-ROM e qualsiasi drive supportato dal sistema, anche future periferiche.

Benché il primo sistema possa sembrare più accattivante per uno che vuole programmare a livello hardware, in realtà si tratta di un modo **vecchio, spesso incompatibile** e limitante per l'impossibilità di installare su HD il programma (o demo che sia). Finché parliamo di una demo o di un gioco per Amiga500 che sia ad 1 solo disco, forse è accettabile l'opzione del trackloader, ma da 2 dischi in sù il sistema porta solo a far arrabbiare i possessori di HardDisk, che saranno sempre di più. Un gioco installato su HD caricherà sempre più velocemente di uno da disco con il loader più turbo possibile.

Poi c'è il discorso della FAST RAM: per poterla utilizzare con un trackloader occorrerebbe fare un mini sistema operativo che trovi dove si trova e rilochi il codice al giusto indirizzo. Non ho intenzione di proporvi il listato di uno di questi loader + rilocatori, per non indurvi sulla cattiva strada. Pensate alla soddisfazione di poter convertire il vostro gioco per il CD32, o di vederlo funzionare sul 68060 e su qualsiasi HardDisk, e invece alla delusione di veder fallire il rilocatore "a mano" o di notare che il programma non sfrutta la fast ram... vi ho convinto?

C'è anche un'altra cosa: sarebbe bene fare uso del comando `assign` per le nostre produzioni che devono caricare files. Ad esempio, se facciamo un gioco ad un disco, dando il nome "Cane" al disco, potremo caricare i file con `Cane:file1`, `Cane:file2`, `Cane2/oggetti/ogg1`, e così via. Nel caso si voglia installare su Hard Disk, basterà fare una directory, copiarci il contenuto del disco, e aggiungere alla `startup-sequence`:

```
assign  Cane: dh0:giococane     ; ad esempio...
```

Se il gioco è a più dischi, basterà copiare tutti i dischi nella directory e fare l'assign di ogni disco:

```
assign  Cane1: dh0:giococane
assign  Cane2: dh0:giococane
assign  Cane3: dh0:giococane
```

Per aggiungere "automaticamente" alla `startup-sequence` o all'`user-startup` gli assign necessari, durante l'installazione del gioco, si possono usare le opzioni dell'installer commodore o altri sistemi, ma questo esula dal corso.

Bene, vediamo allora come caricare un file "path:xx" in una destinazione in memoria. Ci sono vari modi. Il più semplice è questo:

```
1  CaricaFile:
2          move.l  #filename,d1     ; indirizzo con stringa "file name + path"
3          MOVE.L  #$3ED,D2         ; AccessMode: MODE_OLDFILE - File che esiste
4                                   ; già, e che quindi potremo leggere.
5          MOVE.L  DosBase(PC),A6
6          JSR     -$1E(A6)         ; LVOOpen - "Apri" il file
7          MOVE.L  D0,FileHandle    ; Salva il suo handle
8          BEQ.S   ErrorOpen        ; Se d0 = 0 allora c'è un errore!
9
10         MOVE.L  D0,D1            ; FileHandle in d1 per il Read
11         MOVE.L  #buffer,D2       ; Indirizzo Destinazione in d2
12         MOVE.L  #42240,D3        ; Lunghezza del file (ESATTA!)
13         MOVE.L  DosBase(PC),A6
14         JSR     -$2A(A6)         ; LVORead - leggi il file e copialo nel buffer
15
16         MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
17         MOVE.L  DosBase(PC),A6
18         JSR     -$24(A6)         ; LVOClose - chiudi il file.
19 ErrorOpen:
20         rts
21
22
23 FileHandle:
24         dc.l    0
25
26 ; Stringa di testo, da terminare con uno 0, a cui dovrà puntare d1 prima di
27 ; fare l'OPEN della dos.lib. Conviene mettere l'intero path.
28
29 Filename:
30         dc.b    "Assembler3:sorgenti7/amiet.raw",0      ; path+nomefile
31         even
```

Questo è perfetto se si conosce la lunghezza esatta del file da caricare. Trattandosi del nostro programma, si suppone che si sappia quando sono lunghi i nostri file dati! Vediamo un esempio in `Lezione11o1.s`.

Però a noi interessa maggiormente caricare un file mentre stiamo visualizzando una nostra copperlist, e magari suonando una musica in interrupt. Come si concilia un caricamento "legale" con un sistema operativo del tutto disabilitato? Intanto consideriamo il fatto che gli interrupt di sistema devono essere tutti riattivati, mentre la copperlist di sistema non serve, e possiamo tenere la nostra. Allora come continuare a suonare la musica, o fare qualcos'altro, mentre sta avvenendo un caricamento? I sistemi sono molteplici. Potremmo aggiungere delle nostre routines in modo "legale" all'interrupt di sistema, con un `AddIntServer()`. Oppure potremmo eseguire un nostro interrupt, che poi *salti* ad eseguire quello di sistema.

Un modo un pò meno rispettoso, che però funziona e preferisco usare, anche perché l'ho visto usare nei giochi per CD32. In pratica ecco cosa dobbiamo fare: ripristinare i vecchi interrupt e il vecchio stato `DMA/INTENA`, riabilitare il multitasking eccetera, come facciamo all'uscita, ma lasciare la nostra copperlist e "infilare" il nostro interrupt $6c in più a quello di sistema. Poi caricare il file, e **attendere qualche secondo per essere sicuri che la spia del drive o dell'hard disk o del cd-rom si sia spenta**, poi richiudere tutto e tornare a battere nel metallo senza pietà.

Insomma, prima e dopo il caricamento occorre riabilitare e ridisabilitare il sistema operativo, lasciando la nostra copperlist. L'unico particolare è l'interrupt: come facciamo ad eseguire il nostro, poi saltare a quello vecchio? Voglio proporvi un sistema da veri contrabbandieri, che però funziona, a patto che si chiami la routine `ClearMyCache`, che azzera l'instruction cache del processore (68020+). Infatti useremo per la prima (e ultima) volta, codice **automodificante**! Non andrebbe mai usato, ma voglio farvi vedere uno dei pochi casi in cui funziona ed è utile, giusto per informazione. Avete presente che ogni istruzione quando viene assemblata diventa una serie di valori esadecimali? ad esempio `RTS` diventa $4e75, e così via. Noi dobbiamo *jumpare* al vecchio interrupt, dopo aver eseguito il nostro. Dunque, un `JMP $12345`, ad esempio, diventa `$49f900012345`, ossia `$4ef9`, seguito dall'indirizzo a cui saltare, che è una long:

```
        dc.w    $4ef9           ; val esadecimale di JMP
Crappyint:
        dc.l    0               ; Indirizzo dove Jumpare, da AUTOMODIFICARE...
```

Ora, se mettessimo in `CrappyInt` l'indirizzo dell'interrupt di sistema con:

```
        move.l  oldint6c(PC),crappyint  ; Per DOS LOAD - salteremo all'oldint
```

Avremmo il `JMP oldint6c` che cercavamo... allora l'interrupt finale è:

```
*****************************************************************************
; Routine di interrupt da mettere durante il caricamento. Le routines che
; saranno messe in questo interrupt saranno eseguite anche durante il
; caricamento, sia che avvenga da floppy disk, da Hard Disk, o CD ROM.
; DA NOTARE CHE STIAMO USANDO L'INTERRUPT COPER, E NON QUELLO VBLANK,
; QUESTO PERCHè DURANTE IL CARICAMENTO DA DISCO, SPECIALMENTE SOTTO KICK 1.3,
; L'INTERRUPT VERTB NON è STABILE, tanto che la musica avrebbe dei sobbalzi.
; Invece, se mettiamo un "$9c,$8010" nella nostra copperlist, siamo sicuri
; che questa routine sarà eseguita una volta sola per fotogramma.
*****************************************************************************

myint6cLoad:
        btst.b  #4,$dff01f      ; INTREQR - il bit 4, COPER, è azzerato?
        beq.s   nointL          ; Se si, non è un "vero" int COPER!
        move.w  #%10000,$dff09c ; Se no, è la volta buona, togliamo il req!
        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_music        ; Suona la musica
        movem.l (SP)+,d0-d7/a0-a6
nointL:
        dc.w    $4ef9           ; val esadecimale di JMP
Crappyint:
        dc.l    0               ; Indirizzo dove Jumpare, da AUTOMODIFICARE...
                                ; ATTENZIONE: il codice automodificante non
                                ; andrebbe usato. Comunque se si chiama un
                                ; ClearMyCache prima e dopo, funziona!
```

Come vedete, basta puntare in *$6c*+`VBR` questo interrupt per eseguire `mt_music` e il vecchio interrupt di sistema, ottenendo la musica+il caricamento in contemporanea. Vediamo un esempio in `Lezione11o2.s`.

A questo punto vi potete immaginare a cosa può servire la routine che blocca l'input di intuition: quando carichiamo un file, riabilitiamo multitasking e interrupts di sistema quindi, anche se viene visualizzata la nostra copperlist, il workbench funziona perfettamente, tanto che se durante un caricamento si muove "alla cieca" il mouse, si può anche azionare qualche menù o clickare qualche icona, o dare dei comandi da tastiera per il cli. Pensate ad un videogiocatore che ha il vizio di muovere e pigiare il mouse durante i caricamenti per non innervosirsi: all'uscita del gioco potrebbe accorgersi di aver clickato l'icona dell'Hard Disk e aver scelto per caso l'opzione format dal menù del WB che non vedeva, e magari premendo accidentalmente la tastiera potrebbe avergli dato un nome osceno, anche. Quindi, anche se una volta disabilitato il sistema operativo chiamare la routine InputOff non è indispensabile, nel caso si carichino files o si facciano altre operazioni è bene che non sia possibile fare danni!

Per terminare la lezione, vediamo come fare a caricare un file di cui non sappiamo a priori la lunghezza, prendendo anche l'occasione per spiegare le routines di `AllocMem` e `FreeMem`. Per sapere la lunghezza di un file, basta eseguire una apposita funzione, detta `Examine`, a patto che si abbia lockato il file. Ciò non è molto difficile, basta fare qualche `JSR` in più. Da notare che `Examine` non fa altro che riempire un buffer lungo $104 bytes con i vari dati del file, ecco un esempio:

```
        cnop    0,4     ; Attenzione! Il FileInfoBlock deve essere allineato
                        ; a longword, non basta che sia ad un indirizzo pari!

fib:
        dcb.b   $104,0  ; Struttura FileInfoBlock: offsets.
                        ; 0 = fib_DiskKey
                        ; 4 = fib_DirEntyType (<0 = file, >0 = directory)
                        ; 8 = FileName (max 30 caratteri, terminato con 0)
                        ; $74 = fib_Protection, $78 = fib_EntryType
                        ; $7c = fib_Size, $80 = fib_NumBlocks
                        ; $84 = fib_Date (3 longs: Days, Minute, Tick)
                        ; $90 = comment (termina con uno 0)
```

Come vedete, all'offset $7c troviamo la lunghezza. Le altre cose non ci interessano... che ce ne facciamo della data o del commento? Comunque, dato che dobbiamo allocare la memoria per il file, la allocheremo anche per il `FileInfoBlock`, in modo da risparmiarci questo `dcb.b $104,0`. Una volta saputa la lunghezza del file, dovremo creare un buffer in memoria lungo quanto il file, per caricarcelo dentro. Questo si fa con `AllocMem`, che richiede in entrata il numero di bytes da allocare e il tipo di memoria, se chip o no, in modo analogo alle `section` con `_C` o no. A differenza delle sections, però, alla fine del programma dobbiamo liberare manualmente tutti i blocchi allocati tramite la funzione `FreeMem`.

11.15 AllocMem

Questa routine di Exec serve per richiedere un blocco di memoria da usare per i nostri scopi. Basta indicare il tipo di memoria richiesto (in pratica se deve essere CHIP ram o no), e la lunghezza in bytes di tale blocco. La routine **alloca** il pezzo di ram libera per il nostro uso esclusivo, in quanto ne prendiamo possesso, dato che il sistema operativo non scriverà più in quel pezzo di memoria, fino a che non glielo "rendiamo", con `FreeMem`. Infatti il sistema multitasking Amiga funziona con questo sistema: ogni programma richiede quanta memoria gli serve tramite `AllocMem`, il sistema operativo gli riserva dei pezzi di ram libera, poi ad un altro programma che carica in multitasking saranno allocate altre parti di ram libera. Per ora abbiamo usato le `SECTION BSS` per gli spazi di memoria azzerata che ci servivano, in quanto sapevamo la loro grandezza in partenza. Ed è meglio usare le `BSS` per i bitplanes o i buffer di grandezza certa, per varie ragioni, come il non dover chiamare routines e il poter mettere le label qua e là nel buffer, a differenza della mem allocata a cui dovremmo accedere per forza tramite offsets dall'inizio del blocco. Nel nostro listato, carichiamo in memoria un file di cui non sappiamo la lunghezza, per cui qua è obbligatorio usare l'`AllocMem`, dopo che abbiamo saputo quanto spazio occuperà il file. Vediamo in dettaglio la funzione:

```
        move.l  Grandezza(PC),d0 ; Grandezza del blocco in bytes
        move.l  TypeOfMem(PC),d1 ; Tipo di Memoria (chip,public...)
        move.l  4.w,a6
        jsr     -$c6(a6)         ; Allocmem
        move.l  d0,FileBuffer    ; Indirizzo inizio del blocco di mem. allocata
        beq.s   FineMem          ; d0=0? Allora errore!
        ...
```

Se non è indispensabile allocare chip mem (ossia se nel buffer allocato non ci andrà né grafica né suono), allocate sempre `MEMF_PUBLIC`, che significa: *memoria fast se c'è, o se proprio non c'è*

allora chip. Ricordo per l'ennesima volta che è bene risparmiare chip mem, e che la FAST memory è più veloce della chip.

All'uscita, in D0 ci sarà l'indirizzo del blocco di memoria richiesto, che tra l'altro sarà allineato a long word (ossia allineato a 32 bit). Se invece D0 = 0, non è stato possibile allocare un blocco di questo tipo! Testate sempre questa cosa, o in caso di fine mem copiereste tutto in $0!!!

Possiamo anche richiedere che la memoria richiesta sia azzerata, basta settare il bit MEMF_CLEAR, il 16 ($10000). Ecco i parametri più utili da mettere in d1, per richiedere i vari tipi di memoria:

```
MEMF_CHIP       =       2       ; Richiesta Chip Ram
MEMF_FAST       =       4       ; Richiesta Fast Ram (non usatelo)
MEMF_PUBLIC     =       1       ; Richiesta Fast, ma se non c'è va bene chip!
```

E, naturalmente, se volete che i blocchi siano azzerati:

```
CHIP            =       $10002
FAST            =       $10004  ; non usatelo...
PUBLIC          =       $10001
```

Vi sconsiglio di richiedere MEMF_FAST, perché la fast non è presente su tutte le macchine. Usate sempre MEMF_PUBLIC, a parte quando la memoria allocata deve essere usata come bitplane, copperlist o audio, ossia MEMF_CHIP. Da notare che la lunghezza del blocco che immettiamo sarà arrotondata dal sistema operativo ad un multiplo dei blocchi (chunk) del sistema. Questo non è un problema per noi, infatti se immettiano 39, probabilmente alloca 40, ma i 39 richiesti ci sono tutti, quindi a noi non interessa. All'uscita dal programma ricordatevi di liberare il blocco di memoria!

11.16 FreeMem

Questa è la routine da chiamare per liberare i blocchi di memoria allocata. è richiesto l'indirizzo del blocco in A1, e la lunghezza in bytes in D0. ATTENZIONE: Se si tenta di liberare un blocco che non era stato allocato veramente, causerete un casino pazzesco con Guru Meditation/soft Failure! Ecco come liberare il blocco di memoria di prima:

```
1       move.l  Grandezza(PC),d0  ; Grandezza del blocco in bytes
2       move.l  FileBuffer(PC),a1 ; Indirizzo del blocco di mem. allocata
3       move.l  4.w,a6
4       jsr     -$d2(a6)          ; FreeMem
```

A questo punto possiamo anche vedere il programma: Lezione11o3.s.

CAPITOLO 12

LEZIONE 12 - LA COMPATIBILITÀ DEL CODICE

In questa lezione leggerete la tecniche per ottenere la *compatibilità del codice*, una cosa importantissima: pensate quanto sia importante che il gioco o la demo che programmate funzioni su tutti i modelli Amiga!!!

Questo non è assolutamente difficile, anche se è risaputo che moltissimi giochi e demo vecchi non funzionano su kick 2.0, 68020, a1200, o addirittura esistono delle cose che funzionano solo su A500 inespanso 1.3, basta mettere la fast ram, o il kickstart nuovo che non funzionano più. Tutti questi problemi derivano da poche cause, le stesse stupide cause, infatti il 99% del codice di un demo o di un gioco che va solo su a500 1.3 funzionerebbe su tutti gli Amiga, se non fosse per quelle 2 o 3 linee di codice *"sporco"* che fanno inchiodare tutto, generalmente al boot.

Personalmente ho sempre cercato di capire tutti questi BUG, ossia errori di programmazione visibili soltanto su macchine superiori al 500 1.3, e molto spesso sono riuscito a *fixare*, ossia a "riparare" dei giochi o delle demo che non funzionavano, semplicemente disassemblando il codice e modificando gli errori ricorrenti. In questo modo ho unito l'utile al dilettevole: da una parte ho fatto funzionare ad amici con A1200 o altri computer il gioco o la demo che tanto piacevano quando avevano sempre il 500 1.3, prima di venderlo, dall'altra mi sono fatto una discreta cultura sulle cause degli "inchiodamenti" con guru spettacolare, e ho constatato che sono sempre i soliti pochi "vizietti" di programmazione, che elencherò.

Per colpa di questi vizietti la maggior parte del software scritto in assembler diretto sull'hardware Amiga si è guadagnato la fama di essere incompatibile e poco sicuro, per cui l'assembler stesso è stato colpevolizzato di essere un linguaggio insicuro, specialmente quello "hardware direct", il "metalbashing".

Tutti questi problemi invece possono essere facilmente evitati, basta *non fare* certe cose, e sicuramente il gioco/demo girerà su tutti gli Amiga. Infatti tutti i listati di questo corso, per esempio, funzionano sia su amiga 500 1.3 che su amiga 4000/040, e naturalmente vanno anche su tutti gli altri computer intermedi, con qualsiasi configurazione.

Naturalmente non posso garantire la compatibilità con ipotetici modelli di amiga con RISC o chipset AAA, ma in tal caso non funzionerebbe NEMMENO UN GIOCO, e tantomeno programmi come il protracker. Spero infatti che sia mantenuta la serie 680x0 e la compatibilità verso il basso con l'ECS (almeno), o ci troveremmo con dei PC, chiamati "Amiga", ma non MSDOS compatibili.

Farei piuttosto dei computer basati su 68060 a 150Mhz, che non ha niente da invidiare ad un RISC, e un "local bus" della chip ram, ossia una velocizzazione dell'accesso a questo tipo di memoria, che sui modelli attuali è troppo lenta (maledetti ingegneri C= dell'ultim'ora).

Ho deciso di trattare questo argomento solo ora, anche perché per poter fare degli errori è necessario almeno saper programmare, dunque non era logico mettere questa lezione prima di aver spiegato le basi della programmazione. Dopo questa lezione forse riuscirete a far funzionare il vostro vecchio gioco incompatibile!

Nel corso avete visto **come bisogna programmare**, con tutti i procedimenti giusti, per cui ignorate le cavolate che venivano fatte anni fa. Ecco una lista degli errori "alla paperissima" che ho trovato in giro per i programmi che non funzionano su tutte le macchine: (test su A4000/040)

12.1 Errori riguardanti COPPERList e Blitter

1. Tra gli errori "rimediabili" troviamo il meno grave, che poi non è un errore nelle vecchie produzioni, dato che non potevano sapere dell'AGA: è quello di "dimenticarsi" di resettare l'AGA con queste 3 istruzioni dopo aver puntato la copperlist: (Non prima!)

```
        lea     $dff000,a5      ; Indirizzo CUSTOM di base in A5 per offsets
        move.l  #copper,$80(a5) ; COP1LC - Punta la copperlist
        move.w  d0,$88(a5)      ; COPJMP1 - fai partire la copperlist

;       disabilitiamo l'AGA:

        move.w  #0,$1fc(a5)     ; reset sprites wide and DISABLE 64 bit burst
        MOVE.W  #$c00,$106(A5)  ; reset AGA palette , sprite resolution
                                ; and double playfield palette
        MOVE.W  #$11,$10c(A5)   ; reset AGA sprite palette
```

   A questo errore si può rimediare dal boot del computer premendo entrambi i tasti del mouse, e selezionando l'emulazione del vecchio chipset. D'altronde i problemi di copper non finiscono qua, infatti basta dimenticarsi di definire uno qualsiasi dei registri COPPER, che l'errore si affaccia! Infatti ho trovato molte copperlist di vecchi demo/giochi che non definivano i moduli, per cui RIMANGONO I VALORI della copperlist di sistema che non si sa mai come possono essere. Infatti l'errore più comune è quello di non mettere i moduli (*$108* e *$10a*), dando per scontato che siano azzerati. QUESTO ERA VERO PER IL KICKSTART 1.3!!! MA DAL 2.0 IL MODULO NON è ZERO!!! Dunque le intro/demo/giochi si vedono a "strisciate", a meno che non si carichi il kickstart 1.3. Lo stesso vale per `DiwStart/DiwStop` eccetera. Ricordatevi SEMPRE di mettere nella copperlist tutti i registri, anche se sono azzerati, per evitare che rimangano i valori incerti del sistema operativo!!!

```
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $8e,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
```

   Ho trovato anche altri errori balordi spulciando per le vecchie copperlist. Alcuni anziché NON SETTARE dei registri, misteriosamente NE SETTAVANO TROPPI!!! Infatti **non bisogna mai accedere ad un registro sconosciuto, o non ancora utilizzato, né bisogna settare o azzerare bit riservati o senza funzione in registri conosciuti**, in questo modo si rischia di attivare strane funzioni in chipset futuri. Per ora l'evoluzione più grossa la abbiamo avuta da ECS ad AGA, e sono più di quanto mi aspettassi gli errori di "settaggio alla cieca". Per esempio, ho trovato questa stranezza nella copperlist di una vecchissima intro *Ackerlight*:

```
1          ....
2          dc.w    $100,$5000      ; BPLCON0
3          dc.w    $0092,$30       ; DDFSTRT
4  ;---->  dc.w    $106,$FE5       ; Perché hanno aggiunto un registro a quel
5                                  ; tempo inesistente? Sull'AGA falsa la palette
6          dc.w    $102,$CC        ; BPLCON1
7          dc.w    $108,$A8        ; BPL1MOD
8          dc.w    $10A,$A8        ; BPL2MOD
9          ....
```

Questo errore è passato inosservato fino a che non sono usciti gli AGA, e badate bene che questo errore è ASSOLUTAMENTE INELIMINABILE, infatti, prima di disassemblare il materiale che non funziona, faccio tutte le prove possibili per intuire il problema: tolgo la fastmemory, carico il kick 1.3, disabilito le cache, la MMU, azzero il VBR eccetera... Questo errore si presentava comunque: i colori erano sbagliati, tutto il resto funzionava. Infatti l'unico modo per correggere questi errori è di scovarli e di toglierli nel codice. Ho sostituito quel `$106,$fe5` con un `$92,$30`, ossia ho replicato la linea precedente, e tutto ha funzionato a meraviglia. Forse il coder non si è nemmeno accorto di quel move, può essere un errore di battitura, forse voleva scrivere `$108` e non `$106`, chissà, ma state attenti a lavorare solo su registri o bit conosciuti, o avrete la delusione di vedere la vostra produzione non funzionare sull'Amiga 9000 dei vostri nipoti, e questo per una sola, stupida linea di copperlist.

State attenti anche alle strutture degli sprite, dato che già da OCS a ECS ci sono dei bit in più nel quarto byte di controllo: tanti sprite che appaiono "sporchi", o allungati fino alla fine dello schermo sono in quello stato SOLO su macchine ECS/AGA, e non su vecchi a500/a2000, perché la routine di gestione dello sprite anziché azzerare quel bit lo lasciava settato. I progettisti consigliano di lasciare AZZERATI i bit non usati dei registri conosciuti, e di NON ACCEDERE assolutamente a quelli non ancora conosciuti. Vi consiglio vivamente di non dimenticarvelo.

2. Non usate l'istruzione `CLR` su registri *$dffXXX*, perché tale istruzione ha un comportamento diverso su processori 68000 e 68020/30/40, infatti su 68000 causa una lettura ed una scrittura, cioè 2 accessi, mentre su 68020/30/40 causa un solo accesso. Per evitare risultati diversi su processori diversi, ricordatevi di accedere in altro modo ai registri di tipo `STROBE` (`COPJMP1` = *$dff080*, `COPJMP2` = *$dff088* ecc.). Un modo può essere un `MOVE.W d0,$dff080`. Ho riscontrato problemi anche quando viene utilizzato il `CLR` su altri registri *$dffXXX*, per esempio *$dff064* (`BLTAMOD`)

```
1  Esempio1:
2          MOVE.W  #0,$DFF088      ; mai fare CLR.W $dff088!
3  oppure:
4          MOVE.W  d0,$dff088
5          ...
6
7  Esempio2:
8          MOVE.W  #0,$DFF064      ; mai fare CLR.W $dff064!
9  oppure:
10         MOVEQ   #0,d0
11         MOVE.W  d0,$dff064
12         ...
```

Per sicurezza, quindi, vi consiglio di accedere tramite registri o valori diretti (`#0,$dffxxx`), mai usare un `CLR` su un registro `$dffxxx`.

3. Nel 1988-1989 era in uso un modo alquanto stupido di puntare le copperlist, che poi si è rivelato incompatibile con le versioni del sistema operativo dal 2.0 in avanti, dato che venivano date per scontate delle strutture di sistema che, naturalmente, non essendo

documentate dalla Commodore, sono cambiate lasciando "i furbi" con i loro sorgenti che non puntavano più le copperlist su A500+ e A600. Purtroppo qualcuno poco esperto di programmazione ha continuato a "rubacchiare" pezzi di listati vecchi che ha trovato in giro, contenenti questo ridicolo codice di puntamento della copperlist, per cui anche alcune demo del 1990-91 richiedono il kickstart 1.3 per poter funzionare a causa du questa leggerezza. Riporto il penoso codice inventato da qualche "furbo" pioniere del coding:

```
1       move.l  4.w,a6          ; execbase
2       move.l  (a6),a6         ; ???
3       move.l  (a6),a6         ; HAHAHA! GFXBASE??? Solo in kick1.3!
4       move.l  $26(a6),OLDCOP  ; HAHAHA! SALVA VECCHIA COPLIST???
5       move.l  #MYCOP,$32(a6)  ; DOPPIO HAHAHAHA! PUNTA COPLIST???
6       ...
```

Questo pezzo di codice purtroppo è comunissimo nei vecchi listati in giro nelle banche dati, e nelle intro (es. quella degli *ORACLE*). Ricordatevi SEMPRE di non commettere il **duplice** errore contenuto in queste 4 linee di **sporchissimo** codice: Innanzitutto il GFXBASE si trova aprendo la `graphics.library` come viene fatto negli esempi del corso, e non certo facendo due volte `move.l (a6),a6`, questo avviene casualmente solo nei kickstart 1.2 e 1.3 per la particolare struttura della vecchia libreria. Il secondo errore è quello di puntare la copperlist mettendo il suo indirizzo nella struttura `GFXBASE` anziché nel registro `$dff080`, questo causa infiniti disastri, puntate sempre la copperlist con un bel:

```
1       MOVE.L  #Copperlist,$dff080     ; COP1LC
2       move.w  d0,$dff088              ; COPJMP1
```

4. Un'usanza della vecchia generazione di coder era anche quella di non aspettare la fine di una blittata prima di farne un'altra. Questo si trova molto facilmente nel codice scritto prima del 1990, ma ci sono alcuni che continuano tuttora a sorvolare le routine di `WaitBlit`. Effettivamente quando gli Amiga avevano solo il 68000 come processore c'erano dei casi in cui, tra una blittata e l'altra, il processore doveva fare tante di quelle operazioni che il blitter aveva già finito. I programmatori di demo (e purtroppo anche di giochi) spesso pensavano che era inutile aspettare il blitter se, anche senza mettere la routine di Wait, il tutto funzionava. Un esempio è il gioco *PANG*...Ma se hanno risparmiato 2 linee di codice nel loro demo/gioco, non solo non hanno aumentato la velocità di esecuzione (non sono certo un paio di `btst #6,$dff002` a rallentare...), ma non hanno considerato che su processori più veloci il tempo tra una blittata e l'altra si è accorciato, dato che è risaputo che il 68020 è più veloce del 68000, per cui il crash è totale.

   Purtroppo il blitter è rimasto, su ECS ed AGA, della stessa lentezza (su a4000 o a1200 accelerato addirittura è più lento del normale!). Per risolvere i guai prodotti da tali "leggerezze" di programmazione, alle volte basta togliere le cache e la fast ram, per cui anche un 68020, se lavora in chip ram con le cache disattivate, alle volte rallenta abbastanza da evitare una blittata sopra un'altra già in esecuzione. La cosa brutta è che, per motivi di sincronizzazione hardware, su computer come A4000 o A1200 accelerati con 68030, il blitter è *più lento che nel vecchio A500*, per cui, anche se riusciamo a rallentare il processore come un 68000 base, è il blitter più lento che rende il crash inevitabile e non certo gradito. Tra l'altro tenete presente che anche se ci sono tutte le routine che aspettano la fine della blittata prima di farne un'altra, il blitter, essendo più lento su A4000, può causare delle "scattosità" orrende in giochi o demo che vanno fluide (a 50 fotogrammi al secondo) su un A500, rendendo nervoso l'incredulo possessore di A4000 che, invece, credeva di veder girare più velocemente il gioco o la demo. Quindi SEMPRE e COMUNQUE aspettate che il blitter abbia finito:

```
        LEA     $dff000,a5
WaitBlit0:
        BTST.B  #6,2(a5)
WaitBlit1:
        BTST.B  #6,2(a5)        ; controlla 2 volte per un errore nell'A1000
        BNE.S   WaitBlit1
```

P.S: Alle volte potete trovare dei `btst #14,$dff002` anziché dei `btst #6,$dff002`, ma l'effetto è lo stesso, dato che viene testato sempre il bit 6. Infatti 6+8 fa 14. Il `BTST` lavora solo su byte e testa comunque il sesto bit. Si preferisce per motivi estetici (e di logica) usare `btst #6` e non `btst #14`!!

Per darvi un'idea di quanto rallenti il blitter su macchine accelerate, considerate che, per esempio, una routine che blittava 14 bob per fotogramma su un A1200 base, ne blittava solo 12 su un A4000 e solo 9 su un A1200 con scheda acceleratrice GVP 030 a 40Mhz!!!! Considerate quindi che, quando è possibile, è meglio usare il processore che il blitter. Inoltre è sempre bene lasciare qualche linea raster "libera", anziché blittare fino all'ultimo millisecondo. Infatti, in quest'ultimo caso, con il rallentamento del blitter su a4000 o a1200 accelerati non ce la farebbe più in un frame, e il tutto diverrebbe mega scattoso.

12.2 Errori riguardanti CIAA/CIAB, tastiera, timers, trackloaders

5. Routines che fanno lampeggiare il led del tasto caps lock non funzionano su A1200 perché ha una tastiera economica diversa da quelle standard. Tali routines sono presenti in certe demo per "bellezza", eccone una qua di seguito, provatela e noterete il flash su A500/a2000/a3000/a4000, e, invece, un bel reset su un A1200:

```
CAPSLOCK:
        LEA     $BFE000,A2
        MOVEQ   #6,D1           ; bit 6 of $bfee01-input-output bit of $bfec01
        CLR.B   $801(A2)        ; reset TODLO - bit 7-0 of 50-60hz timer
        CLR.B   $C01(A2)        ; CLear the SDR (synchrous serial shitf
                                ; connected to the keyboard)
DOFLASH:
        BSET    D1,$E01(A2)     ; Output
        BCLR    D1,$E01(A2)     ; Input
        CMPI.B  #50,$801(A2)    ; Wait 50 blanks (CIA timer)
        BGE.S   DONE
        BSET    D1,$E01(A2)     ; Output
        BCLR    D1,$E01(A2)     ; Input
        MOVE.W  $DFF01E,D0      ; Intreqr in d0
        ANDI.W  #%00000010,D0   ; checks I/O PORTS
        BEQ.S   DOFLASH
DONE:
        RTS
```

L'amiga 1200 ha un controller della tastiera economico, provate a fare questa prova per verificare: premete il tasto *<r>*, lasciatelo premuto, e premete un'altro tasto, ad esempio la *<u>*. Su un a1200 non succede nulla, mentre su un altro computer la *<u>* appare sullo schermo. Dunque non scherzate con le routine che gestiscono la tastiera! Una delle demo che non funzionano su A1200 per questa routine è ODISSEY.

Per quanto riguarda le routines che "muovono" le testine dei drives, l'errore fondamentale è quello di sbagliare nelle routines di sincronizzazione, facendole con semplici loop "a vuoto" o serie di NOP che, su processori più veloci, vengono eseguite troppo in fretta per attendere abbastanza. Temporizzate col `VBLANK` o col `CIA`!

12.3 Errori riguardanti i processori 68010/20/30/40/60

6. Innanzitutto bisogna ricercare gli errori anche nelle utility che usiamo, e non solo nel nostro eseguibile. Infatti mi è successo spesso di far funzionare demo o intro (che "guravano" subito) semplicemente scompattandole e ricompattandole con un cruncher moderno, ad esempio Powerpacker o lo StoneCracker 4. Infatti molti dei vecchi compattatori (crunchers) ad indirizzi assoluti non funzionano su 68010+, per cui anche se la demo in sè funziona, il solo fatto di essere compattata con un vecchio ByteKiller o TetraPacker la fa andare in guru, prima di partire, durante il decrunch. Come prima cosa, dunque, non compattate il vostro programma con vecchi crunchers, usate Stone Cracker4, PowerPacker o Titan Cruncher. Inoltre è sempre meglio fare codice rilocabile, che codice ad indirizzi assoluti!!!

7. Errori di indirizzo: Alcune vecchie produzioni contengono degli accessi agli indirizzi della ROM, ad esempio:

```
1       JSR     $fce220
```

   Ebbene, il kickstart 1.2/1.3 è localizzato, nei vecchi Amiga, alle locazioni di memoria *$fc0000*, fino a *$ffffff*, per un totale di 256k. è ovvio che in questi kickstart ogni routine ha il suo indirizzo: come abbiamo già visto esiste una "tabella di JMP" all'indirizzo di Execbase, ossia sappiamo che, ad esempio, avendo l'execbase in `a6`, troveremo, *$84* bytes prima, il `JMP` che salta in ROM ad eseguire il Forbid:

```
1       jsr     -$84(a6)        ; Forbid, disabilita multitask
```

   per esempio nel kickstart 3.0 (Version 39.106) questa è la tabella di JMP dell'execbase (una sua parte disassemblata):

```
1       ...
2       JMP     $00F815CC       ; ...
3       JMP     $00F815A2       ; -$96(a6)
4       JMP     $00F81586       ; -$90(a6)
5       JMP     $00F8286C       ; -$8a(a6) - routine del permit
6   --> JMP     $00F82864       ; -$84(a6) - Routine del FORBID
7       JMP     $00F817F8       ; -$7e(a6)
8       JMP     $00F817EA       ; ...
9       ...
```

   Su un computer con kickstart V39.106, si può ottenere un FORBID con un:

```
1       JSR     $F82864         ; Forbid su kickstart V39.106 di A1200/A4000
```

   Ma se, ad esempio, il kickstart V39.106 è caricato via software e non è in ROM, la tabella dei JMP punterà agli indirizzi della RAM dove è stato caricato il kick. Dunque MAI e poi MAI accedere al kickstart in questo modo, o la vostra produzione funzionerà solo sul vostro computer. Con questo esempio, comunque, potete intuire che si possono modificare le tabelle dei JMP sostituendo l'indirizzo in ROM con uno nostro, in modo da far eseguire nostre routines modificate. È in questo modo che agiscono i programmi che modificano il sistema operativo, ad esempio le utility che aggiungono un gadget alle finestre o che aumentano opzioni al workbench. È per questa *"relatività"* del sistema operativo che MAI bisogna saltare nella ROM. L'unico indirizzo fisso del sistema operativo Amiga è *$0004*, ossia l'EXECBASE, che contiene l'indirizzo da cui fare gli offset. Per cui, se volete avere a che fare col sistema operativo, seguite sempre le indicazioni standard. E anche se non ci volete avere a che fare! Questo tipo di errori è letale, tanto che molte vecchissime demo fatte su Amiga500 Kick1.2, non funzionano su Amiga500 Kick1.3, o superiori, neppure caricando via software il kickstart 1.2.

Altri errori di indirizzo, di poco meno gravi, sono quelli che danno per scontato che la fast ram sia a *$c00000*. Originariamente Amiga aveva 512k di CHIP RAM, successivamente si diffuse l'espansione interna che la portava a 1MB, ed è noto che i 512k di fast ram aggiuntivi sono da *$c00000* a *$c80000*. Anche le demo e i giochi allora cominciarono ad essere progettati per riempire l'intero megabyte di memoria, e siccome in quel periodo la grande maggioranza dei programmi era ad indirizzi assoluti, i coder pensarono di assemblare il programma in fast ram, nelle locazioni da *$c00000* a *$c80000*, e di caricare la grafica e la musica in chip ram, da *$00000* a *$80000*. Per cui il programma, oltre ad essere scompattato negli indirizzi assoluti *$c00000*, avevano le istruzioni allocate per quella zona:

```
1       ...
2       MOVE.L  #$c23b40,d0
3       jsr     $c32100
4       ...
```

Queste demo o giochi funzionarono sugli A500 espansi internamente con la classica scheda, ma quando uscirono gli A500 plus, dotati sempre di 1MB di memoria, ma di sola CHIP, tutti questi programmi risultarono inutilizzabili. Questo è avvenuto perché con 1MB di chip, la memoria è disposta in questo modo: i primi 512k si trovano sempre da *$00000* a *$80000*, ma i secondi 512k sono di seguito da *$80000* a *$100000*!!! Per cui un `JSR $c32100` non porta da nessuna parte, comunque porta di sicuro ad un crash spettacolare con i fuochi d'artificio sullo schermo. In seguito a questo, i giochi e le demo successive hanno usato metodi diversi per sfruttare la memoria oltre i primi 512k. Uno di questi è quello di abbandonare del tutto l'indirizzamento assoluto, dimenticando i comandi ORG e LOAD, e anche i programmi in autoboot, dato che quelli, con un loro loader, devono per forza essere messi ad indirizzi assoluti. Se da una parte molti demo/giochi successivi caricabili via DOS divennero rilocabili tramite le SECTIONS al 100%, senza parti caricate ad indirizzi fissi, altri non vollero rinunciare all'autoboot e agli indirizzi fissi, per usare fino all'ultimo byte di memoria. Questi ultimi risolsero il problema in un paio di modi: uno è quello di assemblare due programmi principali, uno fixato con ORG e LOAD a *$c00000*, se veniva accertato che il computer aveva mezzo mega di CHIP e mezzo di FAST, e un altro fixato all'indirizzo *$80000*, da caricare invece nel caso che il computer avesse 1MB o più di CHIP. In questo modo al boot una routine controlla in quale caso siamo, e carica l'uno o l'altro programma principale all'indirizzo giusto, mentre i dati come grafica e suoni sono poi caricati in seguito dal programma principale. Questo sistema ha lo svantaggio di richiede lo spreco di spazio su disco per le due versioni del programma principale. Altri più "bravi" si sono invece programmati un piccolo sistema operativo, il quale al boot prende nota di quali segmenti di memoria sono presenti nel computer, e tramite una routine di allocazione propria riallocano le varie parti del programma all'indirizzo dove trovano la FAST RAM. Questo è sicuramente il miglior modo di fare un programma AUTOBOOT, anche se è piuttosto difficile, e i vantaggi sono questi: immaginate di caricare su un A4000 una demo o un gioco col sistema dei due programmi principali: al boot il programma riconosce la memoria e, avvedendosi che non è presente la memoria a *$c00000*, carica in CHIP RAM a *$80000* il codice. Lo stesso demo/gioco, invece, viene modificato per caricare col mini sistema operativo: al boot questo riconosce che ci sono due blocchi di memoria, quella CHIP da *$000000* a *$200000* e quella FAST da *$7c00000* a *$7fffff*, di conseguenza riloca tutte le parti di codice in FAST RAM e carica la grafica e il suono in CHIP RAM; come è noto il codice in FAST RAM è molto più veloce che in CHIP RAM, specialmente su processori TURBO come il 68040, di conseguenza la demo o il gioco andranno molto più veloci con il codice rilocato in FAST. Però è da notare che coloro che hanno usato loro sistemini operativi hanno visto andare in crash la loro demo con l'avvento dei 68040, o anche con l'avvento

del semplice 68020, perché la motorola garantisce la compatibilità completa verso il basso SOLO in modo user, e non in modo supervisore: infatti il 68040 ha proprie istruzioni per il modo supervisore, e anche il 68060 è compatibile 100% solo in usermode... Figuriamoci processori o computer del futuro, che magari emuleranno il 680x0... **non andate mai in supervisor e non fatevi sistemini operativi**, per darvi un'idea, la bellissima demo WOC 92 dei Sanity, a causa del suo sistemino operativo, non funziona su 68040... e lo stesso è successo alla demo italiana *IT CAN'T BE DONE*, di un mio amico, e in questo ultimo caso sono stato io a trovargli l'errore: la routine supervisor!!! Tutto sommato, penso che sia più facile e SICURO usare le SECTIONS per fare codice eseguibile, anche perché c'è il vantaggio di poterlo installare su HardDisk, e sicuramente nei prossimi anni Amiga dovrà sempre più rendersi competitivo con l'MSDOS, e con questo intendo che il computer di BASE dovrà avere l'HardDisk e la FAST RAM, altrimenti rimarremo in pochi a vedere i giochetti da 1MB che caricano dal dischetto in autoboot, e che non sfruttano nemmeno la velocità del processore non caricando il codice in FAST RAM!!

Fino qua ho indicato come sia meglio fare codice rilocabile, ma cosa succede se usiamo indirizzi assoluti per un file eseguibile caricabile da dos? Ebbene l'introduzione dell'A500+, con 1MB di CHIP ha portato a non far funzionare anche molte delle produzioni a codice *"misto"*, ossia con codice rilocabile, creato con le "SECTION", ma con l'uso di buffer per la grafica NON allocati tramite `Section BSS` o `AllocMem`, bensì stabiliti arbitrariamente:

```
1       lea     $30000,a0       ; Indirizzo bitplane buffer
2       bsr.s   PrintText       ; Stampa il testo a $30000
```

In questo caso non è il codice ad essere non rilocabile, ma il buffer grafico. Di conseguenza non esiste nemmeno la routine di puntamento dei bitplane in copperlist, perché viene messo direttamente il valore *$30000* dal programmatore: (ORRORE!!!)

```
1       ...
2       dc.w    $e0,$0003       ; bpl0pth
3       dc.w    $e2,$0000       ; bpl0ptl
4       ...
```

Vediamo cosa succede sui computer vecchi, quelli con 512k di CHIP e 512k di FAST: supponendo che la intro abbia una section CODE di 20k e una CHIP di 40k (contenente il FONT dei caratteri e la musica), la prima section viene caricata in FAST e la seconda in CHIP, per cui non si arriva alla locazione *$30000*, ma, mettiamo, a *$2a000*. In questo caso tutto funziona, purché questa intro sia la prima cosa che viene caricata dal dos. Sulle macchine più recenti, con SOLO 1MB o 2MB di chip, non essendoci la FAST MEM, viene caricato tutto in CHIP, sia la section CODE che l'altra, in questo modo gli ultimi Kb di codice (o di musica, grafica ecc.) si trovano oltre l'indirizzo *$30000*. Immaginatevi che bel CRASH avviene quando la routine stampa i caratteri sopra il codice! La disperazione dei coder "leggerini", all'uscita di A500+ e a600, era anche che non riuscivano a correggere i listati su tali computer, in quanto l'ASMONE stesso veniva caricato in CHIP RAM, oltrepassando le locazioni *$30000* o *$40000* usate come buffer assoluti, per cui al JMP poteva anche funzionare (PER CASO) il listato, ma all'uscita l'ASMONE si ritrovava PERFORATO dalle routines e il CRASH era inevitabile. Questo ha insegnato anche ai produttori di intro che bisogna farsi un bel buffer rilocabile:

```
1       SECTION BufferOK,BSS_C
2
3       ds.b    10000
```

Per concludere la serie degli errori riguardanti gli indirizzi, riporto ora degli errori che difficilmente avreste fatto, dato che si tratta di azioni illogiche, ma per sicurezza è bene sapere che:

alcuni coder furboni hanno alle volte usato il byte alto degli indirizzi per scrivere messaggi o semplicemente per il gusto di scriverci.

Dovete sapere che le CPU a 16 bit come il 68000 o il 68010 ignorano il byte alto di un indirizzo, per cui fare:

```
1       JSR     $00005a00
2       JSR     $00120d00
3       JSR     $00c152b0
4       JSR     $00013cd0
```

è equivalente a scrivere

```
1       JSR     $C0005a00
2       JSR     $DE120d00
3       JSR     $FEc152b0
4       JSR     $DE013cd0
```

Si legge chiaramente nei primi byte un "C0DE-FEDE", che può essere un messaggio lasciato da un Emilio Fede coder di tanti anni fa, che si firmava in questo modo. Da notare che con gli esadecimali si possono formare molte parole (A,B,C,D,E,F, e lo 0 come una "O"), ad esempio: FEDE, AFA, ABAC0, FACCE, F0CA, CACCA, CADE, C0DE, ...

Questi furboni dunque lasciavano messaggi interi, poesie, lettere d'amore nei byte alti degli indirizzi nelle serie delle subroutine o in altri luoghi, dove chi disassemblava poteva leggersi anche insulti! Questo giochetto è durato assai poco, per fortuna, ma le intro/demo che li hanno non funzionano su processori a 32 bit, infatti su tali processori l'indirizzamento massimo è aumentato, per cui il JSR cerca veramente quelle strane locazioni. Tra l'altro avrete notato come la FAST RAM dei vecchi A500 sia a *$00c00000*, mentre quella dell'A4000 sia a *$07c00000*, cioè fuori del raggio di indirizzamento di un 68000.

L'ultimo tra gli errori di indirizzo, e non meno insolito di quello precedente, è quello della memoria CHIP da 512k, che viene "ripetuta" quattro volte nel bus indirizzi, nel senso che nonostante che questa si trovi da *$00000* a *$7ffff*, ci si può accedere anche operando su *$80000-$FFFFF*, oppure *$100000-$17ffff*, o *$180000-$1FFFFF*. In pratica i *$80000* bytes (512k), sono 1/4 dei *$200000* (2MB) del bus, e sulle macchine OCS (vecchi amiga che potevano indirizzare solo 512k di CHIP, il resto FAST), ogni byte di memoria CHIP è accessibile da quattro indirizzi diversi, distanziati l'uno dall'altro da 512kb. Questa naturalmente è una proprietà che sulle macchine ECS ed AGA, cioè quelle che possono indirizzare 1MB o più di memoria, si è persa.

Facciamo un esempio: se scriviamo alla locazione *$0* il valore *$12345678*, possiamo "ripescare" quel valore anche da *$0 + $80000*, *$0 + $80000*2*, nonché *$0 + $80000*3* e *$0 + $80000*4*. Vediamo un listato:

```
1       move.l #$12345678,$0    ; mettiamo nei primi 4 bytes questo valore
2
3       move.l  $80000,d0       ; d0 = $12345678
4       move.l  $100000,d1      ; d1 = $12345678
5       move.l  $180000,d2      ; d2 = $12345678
```

Leggendo da *$80000*, *$100000* e *$180000* è come se leggessimo da *$0*!!!! Purtroppo qualche stupido ha usato questa strana proprietà per le sue routines, e questo causa il cattivo funzionamento di a500+ e a600 di varie cose, e badate bene che nonostante che il processore sia sempre il 68000, l'errore si presenta anche con kickstart 1.3 in ROM.

Dunque, ora conoscete tutti gli errori riguardanti gli indirizzi che sono stati fatti nel passato. Vedete di non farli e di non inventarne di nuovi!!!!

8. Problemi con lo SR nei processori 68010 e superiori: Uno dei problemi di incompatibilità più frequenti nei processori 68010 e superiori rispetto al codice 68000 è quello delle istruzioni MOVE SR,dest ad esempio MOVE SR,d0 o MOVE SR,$1234 o MOVE SR,LABEL. Infatti queste istruzioni su 68000 base possono essere usate normalmente in modo utente (MODO USER) come qualsiasi altra istruzione: programmi come gli emulatori (PC Transformer, C64 Emulator ecc.) non funzionano su 68010+ proprio perché eseguono questa operazione in modo USER, il che su 68010+ non è più possibile e causa un guru di *"Privilege Violation"*.

   Anche molti giochi e demo si inchiodano per la presenza di questa istruzione nella parte iniziale delle routines che prendono il controllo del sistema. La Motorola decise di aggiungere ai processori dal 68010 in avanti la possibilità di simulare il funzionamento di nuovi sistemi operativi per macchine ancora non disponibili, questo comportò la necessità di rendere l'istruzione MOVE SR,dest privilegiata, ossia eseguibile soltanto in modo supervisore. Altrimenti il risultato è una GURU di "Privilege Violation". Per accedere allo SR in modo utente, comunque, i progettisti Motorola hanno aggiunto dai processori 68010 in avanti l'istruzione MOVE CCR,dest da usare al posto di MOVE SR,dest, che però non era disponibile su 68000, per cui alcuni programmi per il 68000 di Amiga furono scritti utilizzando l'istruzione MOVE SR,dest in modo utente, ed ora lo verifichiamo quando un gioco o una demo si inchiodano al boot con un sinistro messaggio di GURU MEDITATION o di SOFTWARE FAILURE su A1200/A3000/A4000 o A2000 accelerati.

   In realtà lo "sbaglio" lo ha fatto la Motorola, in quanto gli ignari che hanno usato fiduciosamente MOVE SR,dest in modo utente (USER) non si aspettavano certo che divenisse un'istruzione da eseguire solamente in modo supervisore! (dopo un TRAP o una EXCEPTION del processore).

   Comunque l'importante è saperlo, ed ora possiamo essere sicuri di non incorrere nel problema, e questo è possibile ricordandosi di eseguire sempre l'istruzione MOVE SR,dest in modo SUPERVISORE, ossia dopo un *TRAP*, o in un *INTERRUPT*, eccetera. In questo modo l'istruzione funzionerà su tutti i processori. Un'altra soluzione potrebbe essere quella di controllare che processore c'è sulla macchina, ed eseguire il codice appropriato, ossia un MOVE SR,dest su 68000, oppure un MOVE CCR,dest su 68020, entrambi in modo USER per evitare di doverle eseguire in modo SUPERVISOR, ma ritengo che la soluzione più veloce e ragionevole sia quella di eseguire sempre il MOVE SR,dest in modo SUPERVISOR. Recapitolando:

```
CPU             Modo USER (UTENTE)      Modo SUPERVISORE

68000           MOVE SR,dest            MOVE SR,dest
68010/20/30/40  MOVE CCR,dest           MOVE SR,dest
```

   Converrete che conviene eseguire sempre il vecchio MOVE SR,dest in modo supervisor, ciò risparmia tempo e routines. Se invece il gioco/programma/demo che state programmando è destinato solo a processori 68010+, ad esempio se la demo è solo AGA, potete usare il nuovo MOVE CCR,dest in modo utente, dato che siete su un 68020+, ma ricordatevi anche che tale istruzione non esiste su 68000, per cui non viene assemblata da assemblatori 68000 base, come questo TRASH'M-One; per poter assemblare istruzioni 68010+ come questa dovete usare il TFA ASMONE o il DEVPAC 3.

   D'altronde, vi consiglio proprio di **non usare mai questa istruzione**, e di **non andare mai in modo supervisore**... a che vi serve? Per rischiare? Quando avviene un errore di questo

tipo, il numero del *SOFTWARE FAILURE* da sistema operativo è *#80000008*, non è difficile identificarlo.

Comunque per programmare demo o giochi agire sul registro SR non ha una utilità fondamentale, per cui vi consiglio VIVAMENTE di non accedere MAI a questo registro, anche perché i suoi bit sono diversi da processore a processore, ed è facilissimo causare problemi di incompatibilità.

9. Col 68010, oltre al comando `MOVE CCR,SR` che abbiamo visto, sono state introdotte altre novità, che se non conosciute possono causare errori di incompatibilità. Si tratta del `VBR`, ossia del VECTOR BASE REGISTER, che significa "registro di base del vettore". Abbiamo visto questo registro nella lezione sugli interrupt, infatti sappiamo che quando avviene un interrupt o una trappola (istruzione `TRAP #xx`), il processore INTERROMPE la lettura del programma che stava eseguendo in modo USER, passa al modo SUPERVISORE ed esegue la routine all'indirizzo che trova nel VETTORE specifico, che può essere uno dei livelli di interrupt, oppure uno dei TRAP, eccetera.

   Nei nuovi processori, oltre ad essere stati usati vettori che nel 68000 non avevano funzioni (si vedano, ad esempio, *$18* e *$1c*), è stata implementata la possibilità di spostare la BASE di questi vettori. Mentre su un 68000 siamo sicuri che l'interrupt del `VBLANC` è sempre all'indirizzo *$6c*, su un 68010 o superiori non ne possiamo essere sicuri. Questo perché la base di questi OFFSET può non essere più *$000000*. Infatti basta eseguire, in supervisore, un `MOVEC d0,VBR`, e cambia tutto. Naturalmente al momento del boot il VBR è azzerato, per cui i vettori sono tutti allo stesso posto del 68000. È il *SetPatch* dell'AmigaDos che sposta il VBR, normalmente in FAST RAM, copiando gli indirizzi dei vettori al nuovo indirizzo. Oppure lo "spostamento" viene fatto da altre utility. Di fatto, quindi, una volta caricato il WorkBench su un computer con 68010+ è molto probabile che il VBR non sia a zero, per cui le vecchie demo e i giochi (non solo quelli vecchi!), se caricati dallo Shell o dal WorkBench, spesso non hanno la musica o proprio si inchiodano, perché mettono le routine di interrupt in *$6c*, quando lo dovrebbero mettere in *VBR+$6c*. Quindi MAI si deve fare una cosa del genere:

```
1       MOVE.L  #IntRoutine,$6c
```

   Per prima cosa, si poteva "ottimizzare" in:

```
1       MOVE.L  #IntRoutine,$6c.w
```

   Ma la cosa più importante è che funziona solo se caricata al boot, prima di eseguire il SetPatch o altre utility. Per ovviare al problema, bastano poche righe di codice, che controllino se è presente un 68000 o un 68010+, e in quest'ultimo caso leggano il valore del VBR per eseguire i dovuti offset. Alla fine del programma, basterà ricordarsi di fare lo stesso per rimettere a posto il vecchio interrupt. Questo accorgimento è presente nella `startup2.s` usata nei listati avanzati del corso. Da notare che l'istruzione `MOVEC VBR,A1`, essendo 68010+, non viene assemblata da tutti gli assemblatori (compreso questo ASMONE), per cui è meglio metterlo tramite il suo equivalente esadecimale. Nessuno infatti vi impedisce di scrivere `dc.w $4e75` al posto degli `RTS`!

10. Ora vediamo le leggerezze di programmazione che sono state rese evidenti con l'introduzione delle *INSTRUCTION CACHE* dei processori dal 68020 in avanti. Per "rese evidenti" intendo che tali errori portano ad un crash del sistema spaventoso. Fortunatamente molti di questi errori possono essere risolti disattivando le CACHE tramite utility software.

Vediamo in breve cosa sono queste CACHE: si tratta di memoria molto veloce che si trova DENTRO il processore anziché fuori, al contrario della CHIP o FAST ram, che per essere raggiunte occorre passare dall'autoBUS.

Abbiamo già visto i REGISTRI dati e indirizzi, che non sono altro che LONG di memoria interna al processore, che possiamo leggere e scrivere. Ebbene le CACHE sono banchi di memoria simili, che però non possiamo leggere o scrivere con delle istruzioni, vengono lette e scritte automaticamente dal processore tramite un apposito hardware. Lo scopo delle cache è di velocizzare i LOOP, ossia le routines che vengono eseguite ciclicamente molte volte. Premetto che su 68020 e 68030 l'INSTRUCTION cache è di 256 byte, mentre su 68040 è di 4096 byte. Su 68060 credo sia di 8192, e in futuro chissà... Ebbene, immaginate questo loop:

```
1           ...
2           MOVEQ    #100,d0
3   Loop1:
4           move.w   LABEL1(PC),d2
5           add.w    d3,d2
6           ....
7           altre istruzioni
8           ....
9           DBRA     d0,loop1
10          ...
```

Anche aumentando la velocità del processore, questo loop richiede la lettura delle istruzioni tra la label `loop1:` e il `DBRA d0,loop1` ogni ciclo, e la lettura da RAM, specialmente se è CHIP RAM, è molto lenta. I progettisti Motorola quindi hanno escogitato questo trucco: "e se mettessimo automaticamente nella cache memory gli ultimi 256 byte che sono stati eseguiti?? Otterremmo che quando si presenta un loop più piccolo di 256 byte, tutte le istruzioni del loop stanno nella CACHE e il processore può leggerlo le restanti volte dalla veloce memoria CACHE anziché dalla RAM!". Così più o meno funziona la Instruction CACHE.

Il loop di prima sarebbe letto dalla RAM solo la prima volta poi, raggiunto il DBRA, il processore "si accorge" che `Loop1:` è abbastanza vicino da essere contenuto sempre nella CACHE, e le restanti 99 volte il tempo di lettura delle istruzioni si abbassa notevolmente essendo eseguito dalla CACHE anzichè dalla CHIP/FAST RAM.

Vi chiederete: ma allora quali errori si possono verificare????? Il più comune è quello di chi ha "temporizzato" certe routines basandosi sul tempo che occorre al 68000 base per fare un certo numero di loop "a vuoto", (ora si ritrova con un listato da buttar via). Vediamo i "furbi" come hanno fatto a "perdere del tempo" per aspettare, ad esempio, che le testine del disk drive si spostassero, o per temporizzare una musica, o altro ancora:

```
1           ....
2           MOVE.W   #2500,d0
3   Aspettatempo:
4           dbra     d0,Aspettatempo
5           ...
```

Questi esempi di programmazione impacciata ed approssimativa sono, purtroppo, molto frequenti nei track loaders e nelle routines che suonano le musiche. La routine `music.s` presente nel corso aveva in origine un paio di questi "loop a vuoto", che ho prontamente sostituito con routines "perditempo" affidabili, che ora vedremo.

Se avete delle replay routines per noisetracker/protracker quasi sicuramente troverete dei loop stupidi di questo tipo, che causano la perdita di alcune note durante l'ascolto della martoriata musica. Una nota: anche nei listati del corso di Gerardo Proia ci sono loop stupidi di questo tipo, spero non li abbiate assunti come esempio!

Cercate nei listati che avete trovato in giro questi loop maledetti, e se li trovate buttate via quei sorgenti schifosi o sostituite quelle parti con routines che usino il CIA o il VBLANK, per temporizzare. Il principio di funzionamento di un loop a vuoto è che il processore deve leggere, in questo caso 2500 volte, l'istruzione DBRA dalla memoria, sottrarre #1 a d0 e saltare indietro ad Aspettiamo:.

Su un computer con la cache attiva si può far leggere anche 50000 volte, ma, essendo finito in CACHE, il DBRA verrà eseguito in una frazione di secondo comunque, di conseguenza il drive non legge le tracce, e la musica "taglia" delle note. Oltretutto anche il 68010, per la verità, ha una piccola CACHE di 3 word per velocizzare i piccoli loop DBRA come questo, per cui tali loop "funzionano" solo su 68000 a 7Mhz. Dato che nel corso MAI viene insegnato a temporizzare con loop a vuoto, spero che nessuno cominci autonomamente a fare simili **CAZZATE**.

In generale si può dire che MAI bisogna prendere come riferimento la velocità di esecuzione delle istruzioni da parte della CPU 680x0, dato che varia da processore a processore, e addirittura a seconda di quale tipo di memoria viene letta. Le uniche certezze, dal punto di vista della temporizzazione, sono **il refresh video del VBLANK**, che in standard PAL sarà sempre eseguito 50 volte al secondo, e **i timer del CIA**, che è un chip uguale per tutti gli Amiga, per cui 1 millisecondo è uguale su un A500 come su un A4000. Badate di non basarvi neppure sulla velocità del blitter, perché varia di velocità a seconda del processore del computer.

Ecco come temporizzare trackloaders, segnali per le periferiche esterne o bracci robot collegati alla porta parallela:

Innanzitutto vediamo come aspettare qualche "linea raster" col VBLANK:

```
1         LEA     $DFF000,A5      ; Custom register base in a5
2  PerdiTempo:
3         MOVE.w  #LINEE-1,D1     ; NUMERO DI LINEE da ASPETTARE (304=1 frame)
4  VBWAITY:
5         MOVE.B  6(A5),D0        ; $dff006, VHPOSR.
6  WBLAN1:
7         CMP.B   6(A5),D0        ; VHPOSR
8         BEQ.S   WBLAN1
9  WBLAN2:
10        DBRA    D1,VBWAITY
```

Se non state usando i timer del CIA per altre routines, allora potreste usarli, anche se è meglio toccarli il meno possibile, dato che li usa il sistema operativo. Nella lezione11 trovate sorgenti sull'argomento. Nell'usare i timer del CIA considerate che anche il sistema operativo ne utilizza alcuni per certi scopi: (meglio usare il CIAB!)

```
   CIAA, timer A        Utilizzato per l'interfacciamento con la tastiera

   CIAA, timer B        Utilizzato dall'exec per lo scambio dei task ecc.

   CIAA, TOD            timer a 50/60 Hz utilizzato dal Timer.device

*  CIAB, timer A        Non utilizzato, disponibile per i programmi

*  CIAB, timer B        Non utilizzato, disponibile per i programmi

   CIAB, TOD            Utilizzato dalla graphics.library per seguire le
                        posizioni del pennello elettronico.
```

Se dovete usare timer che servono anche al sistema operativo, fatelo solo se avete disabilitato multitasking e interrupt di sistema, se avete cioè preso il controllo completo del

sistema. Comunque è sempre più pericoloso usare il CIA che il VBLANK, perché all'uscita dalla nostra produzione se abbiamo sballato un timer chissà cosa potrebbe succedere.

Oltre al problema dei loop di temporizzazione, c'è anche quello dato da un altro vizio di programmazione, che però al giorno d'oggi è quasi scomparso, fortunatamente. si tratta del leggendario e misterioso codice *"automodificante"*: questo tipo di codice è detto automodificante proprio perché modifica se stesso. Infatti è possibile fare delle "creature" che, oltre a modificare dei dati, modificano anche le proprie istruzioni durante l'esecuzione.

Questo tipo di programmazione purtroppo è stato usato fin dall'antichità, probabilmente perché sembrava un modo per scrivere codice più veloce o più potente. In realtà quello che viene fatto con codice automodificante può essere riscritto con codice normale al 100% e, alle volte, si può guadagnare in velocità. Per cui dimenticatevi di scrivere codice di questo tipo, se non proprio a scopo di esperimento, perché tale codice non funziona con le CACHE del 68020/30/40/60 attive.

Chi ha un A1200 si rende certamente conto di quanti siano i giochi e le demo che non funzionano solo a causa delle cache! In effetti, credo che la maggior parte degli errori nel vecchio software sia di questo tipo. Per riconoscere un gioco o una demo che ha codice automodificante, basta provare se funziona togliendo le cache (senza caricare vecchi kick), poi riprovarla con le cache attivate. Se a questo punto non funziona, è ovvio che sono solo le cache attive a causare il problema, e le cache causano solo due tipi di errore: quello dell'annullamento dei cicli di ritardo DBRA, che abbiamo già visto, e quelli dovuti al codice automodificante. Ecco come si può presentare un listato contenente codice automodificante:

```
1           ...
2           divu.w  #3,d0
3   MYLABEL:
4           moveq   #0,d0
5           ...
```

Che viene assemblato in memoria in questo modo:

```
1           ...
2           dc.l    $80FC0003       ; DIVU.W #$0003,D0
3   MYLABEL:
4           dc.w    $7000           ; MOVEQ  #$00,D0
5           ...
```

Per ora abbiamo modificato dei dati in memoria, nelle copperlist, abbiamo messo valori nei registri custom *$dffXXX*, ma non abbiamo mai agito DENTRO una istruzione!!! Questo proprio perché è una cosa **incompatibile**. (In realtà abbiamo modificato un JMP alla fine di un interrupt nel listato sul caricamento da dos in Lezione11.txt, ma è l'unico caso "utile"!). Immaginatevi che, più avanti nel listato, ci sia questa istruzione:

```
1           ...
2           move.w  #5,MYLABEL-2
3           ...
```

Cosa succede? La word che si trova prima della label MYLABEL è il *$0003* del DIVU #3,d0, che diventa *$0005*, per cui il DIVU #3,d0 diventa DIVU #5,d0!!! Allo stesso modo si possono modificare tutte le altre istruzioni. In altro modo si può scrivere:

```
1           ...
2           divu.w  #3,d0
3   MYLABEL:    EQU     *-2
4           moveq   #0,d0
5           ...
```

Ora per modificare il numero del DIVU basta un MOVE.W #xxxx,MYLABEL, infatti tramite l'EQU *-2 si fa corrispondere MYLABEL con la label -2. L'asterisco si può tradurre in "questo punto", per cui *-2 diventa "questo punto meno 2 bytes". Supponiamo ora di avere questa situazione in un listato con codice automodificante:

```
        ...
        divu.w  #0,d0    ; da modificare nel numero voluto
MYLABEL:
        EQU     *-2
        ...
```

Immaginate cosa accade quando la ICACHE è attiva: nella cache va l'istruzione così come è in memoria, ossia DIVU.W #0,d0, probabilmente prima che sia modificata, per cui al momento che viene eseguito il:

```
        move.w  #5,MYLABEL
```

Viene modificata in RAM l'istruzione DIVU, ma non in CACHE! Infatti tale istruzione verrà eseguita così come era, ossia DIVU.W #0,d0, il che causa un bel crash di sistema per DIVISIONE PER ZERO!!

```
        divu.w  #0,d0    ; il valore "0" sarà sostituito prima che questa
MYLABEL:                 ; istruzione sia eseguita, ma con la ICACHE attiva
        EQU     *-2      ; questa istruzione sarà letta dalla cache come era
                         ; originariamente, e un bel guru DIVISION BY ZERO
                         ; fermerà il nostro divertimento.
```

Allo stesso modo un:

```
        JMP     0        ; l'indirizzo sarà messo in questo punto da un move,
MYLABEL:                 ; ma con la cache avremo un bel JUMP a 0!! (guru!!)
        EQU     *-6      ; (EQU *-4, EQU *-8, a seconda della grandezza
                         ; dell'indirizzo o della "modifica" stupida.
```

In alcuni casi anziché un vero e proprio GURU si verificano dei problemi di malfunzionamento delle routines, per esempio un loop di questo tipo:

```
        ...
        MOVE.W  #100,d0
Loop1:
        ...
        divu.w  #2,d2
MYLABEL:
        EQU     *-2
        ....
        addq.w  #1,MYLABEL
        DBRA    d0,loop1
        ...
```

Non si verifica alcun crash del sistema, ma mentre il divu.w ogni ciclo dovrebbe cambiare in DIVU.W #3,d2, DIVU.W #4,d2 eccetera, invece rimane sempre DIVU.W #2,d2 (essendo letto dalla CACHE e non dalla RAM). Se per esempio questa era una routine che faceva muovere un solido 3d, state certi che il solido rimarrà fermo o non si visualizzerà nemmeno.

Come avremo dovuto fare. La stessa cosa si sarebbe potuta fare dividendo per una LABEL o per un REGISTRO anziché per un valore assoluto (divu.w LABEL(PC),d2 anziché divu.w #xxx,d2), e si sarebbe potuto fare poi l'ADD sulla label, mantenendo la compatibilità con le cache e senza perdere in velocità.

Dunque **non siate stupidi**: fare codice automodificante non è una cosa di cui vantarsi, perché non è né difficile né utile, bensì è incompatibile. Vi prego di non usare vecchi listati che hanno codice automodificante. Esiste comunque un modo, oltre a quello di

disabilitare le cache, per far funzionare codice automodificante: si può infatti AZZERARE, ossia PULIRE la cache con una apposita istruzione, in questo modo verrà cancellata la vecchia istruzione dalla CACHE e il processore dovrà leggere quella modificata dalla RAM. Se volete fare programmi "batterici" o di intelligenza artificiale o chissà di che tipo, che si automodifichino, basta mettere un `BSR.w CACHECLR` prima dell'esecuzione dell'istruzione modificata. Su kickstart 2.0+ esiste un'apposita funzione della ExecLib:

```
ClearMyCache:
        movem.l d0-d7/a0-a6,-(SP)
        move.l  4.w,a6
        MOVE.W  $14(A6),D0      ; lib version
        CMP.W   #37,D0          ; è V37+? (kick 2.0+)
        blo.s   nocaches        ; Se kick1.3, il problema è che non può
                                ; nemmeno sapere se è un 68040, per cui
                                ; è rischioso.. e si spera che uno
                                ; stupido che ha un 68020+ su un kick1.3
                                ; abbia anche le caches disabilitate!
        jsr     -$27c(a6)       ; cache cleaR U (per load, modifiche ecc.)
nocaches:
        movem.l (sp)+,d0-d7/a0-a6
        rts
```

Nella `startup2.s` è presente questa subroutine, che può essere eseguita. Nota: se siamo su kick 1.3 esce senza fare il JSR, in questo modo non ci sono problemi su computer vecchi.

Eseguite questa routine anche dopo aver caricato con un trackloader dei dati in memoria, o dopo aver fatto altre modifiche a zone di memoria contenente codice. Fate attenzione che alle volte un programma con codice automodificante può funzionare per caso su 68020/68030, perché ci sono più di 256byte di distanza tra l'istruzione modificata e quella che modifica, per cui l'istruzione viene letta dalla RAM, ma su A4000 la cache è di 4096 bytes, e chissà su processori futuri quanto può essere aumentata! Dunque MAI cadere nell'automodifica, come invece continuano a fare alcuni coder di demo per a1200, che ottengono solo di non far funzionare le loro creature su a4000.

Nei processori 68030 e 68040 esiste anche la *DATA CACHE*, che fa la stessa cosa della INSTRUCTION CACHE, ma sui dati (tipo le tabelle), ma solo se tali dati sono in FAST RAM. Su CHIP RAM la DATA cache non agisce. Errori per la DATA CACHE sono meno frequenti, infatti è difficile che vengano cambiate, ad esempio, delle tabelle. Comunque per sicurezza potete fare un CACHECLR, anche perché su 68040 è presente il *copyback*, un "potenziamento" della DATA CACHE che è veramente malvagio, tanto che non funzionano nemmeno alcuni programmi compilati in linguaggio C. Per questo ogni tanto fatevi un "ClearMyCache", che non fa male.

11. Un altro problema di incompatibilità che ho riscontrato è quello degli interrupt su A4000. Notavo che molte demo, anche AGA, che funzionavano benissimo su A1200, su A4000 suonavano la musica a doppia velocità, andavano a scatti e a volte inchiodando il sistema. Mi sono reso conto che questo avveniva per una dimenticanza, che come tutte le dimenticanze si fanno notare quando il codice gira su A4000. Date un occhiata a questo interrupt di livello 3 (il *$6c* per intenderci):

```
INTERRUPT:
        MOVEM.L D0-D7/A0-A6,-(SP)
        BSR.W   ROUTINE1
        BSR.W   ROUTINE2
        BSR.W   ROUTINE3
        BSR.W   ROUTINE4
        MOVEM.L (SP)+,D0-D7/A0-A6
NOINT:
        move.w  #$20,$dff09c    ; INTREQ - vertb (bit 5 - $20 = %100000)
        rte
```

Cosa è che non va?? Badate che è stato messo bene in VBR+$6c, per cui viene eseguito regolarmente. La soluzione è: **manca il test del bit in** INTREQR!!!! Ecco come deve essere modificato:

```
1  INTERRUPT:
2          btst.b  #5,$dff01f      ; INTREQR - vertb int? (bit5)
3          beq.s   NOINT           ; non un vero interrupt VERTB
4          MOVEM.L D0-D7/A0-A6,-(SP)
5          BSR.W   ROUTINE1
6          BSR.W   ROUTINE2
7          BSR.W   ROUTINE3
8          BSR.W   ROUTINE4
9          MOVEM.L (SP)+,D0-D7/A0-A6
10 NOINT:
11         move.w  #$20,$dff09c    ; INTREQ - vertb (bit 5 - $20 = %100000)
12         rte
```

Effettivamente molto spesso su a500/a1200 l'interrupt funziona bene anche senza il controllo del bit in INTREQR, ma su A4000 va SEMPRE messo, altrimenti l'interrupt viene eseguito un miliardo di volte di troppo. Quindi MAI e poi MAI dimenticarsi di testare i bit che hanno generato un interrupt in INTREQR! Come abbiamo già visto, per ogni livello di interrupt c'è un bit del *$dff01f* (INTREQR) da testare.

Un promemoria:

```
INT $64      LEVEL1  bits 0 (soft) ,1 (dskblk) ,2 (serial port tbe)

INT $68      LEVEL2  bit 3 (ports)

INT $6c      LEVEL3  bits 4 (copper) ,5 (verticalblank) ,6 (blitter)

INT $70      LEVEL4  bits 7 (aud0) ,8 (aud1) ,9 (aud2) ,10 (aud3)

INT $74      LEVEL5  bits 11 (serial port rbf) ,12 (disksyn)

INT $78      LEVEL6  bit 13 (external int)
```

Ricordatevi di fare sempre il btst all'inizio dell'interrupt, e in caso il bit non sia settato (beq) uscite senza eseguire le routines. All'uscita bisogna sempre agire sul *$dff09c* per rimuovere la richiesta di interrupt, in pratica "segnamo" che l'interrupt è stato eseguito.

State attenti a non commettere l'errore, che pure io ho fatto in passato, di fare, ad esempio, un BTST #11,$dff01f. In questo caso in realtà si testa il bit 11 di 8, ossia il bit 3 del *$dff01f*. Ricorderete la vicenda del waitblit, per cui scrivere btst #6,$dff002 è uguale a scrivere btst #14,$dff002. Alcuni assemblatori infatti assemblano anche istruzioni BTST su indirizzi con numero di bit superiore a 7, nonostante che siano inutili dato che scala di 8 il numero di tale bit. Altri assemblatori, come il Devpac 3, danno errore e non permettono di assemblare tali inutili BTST.b. (NOTARE il .BYTE! = da 0 fino a 7 max!)

GUAI A CHI FA UNO DEGLI ERRORI DESCRITTI NELLA LEZIONE!!!!!!!!!!!

CAPITOLO 13

LEZIONE 13 - OTTIMIZZAZIONE DEL CODICE ASSEMBLY

Autori: Fabio Ciucci, Ugo Erra
Ringraziamenti: Michael Glew, 2-Cool/LSD, Subhuman/Epsilon

Scrivere routine in assembly non significa necessariamente che il proprio codice girerà al massimo della velocità. Infatti non sempre il codice assembly può essere classificato come il meglio ottenibile in termini di velocità. Consideriamo infatti i numerosi demo che esistono in circolazione ed esattamente quelli che trattano grafica 3d, nella maggior parte delle volte (quasi sempre) le routine che stanno sotto ad effetti quali rotazioni, zoom, esplorazioni di mondi, etc, sono le stesse, ma la loro implementazione in codice assembly è differente, poiché ogni programmatore cerca di implementarle nel modo migliore possibile cioè in modo che girino al massimo della velocità. Ciò è realizzato con tecniche di ottimizzazione che ogni buon coder assembly deve sapere. Le tecniche sono numerose e sicuramente ci vuole un bel pò di tempo prima che si inizino ad utilizzare in modo del tutto naturale. Esistono vari tipi di ottimizzazione e molte di queste tecniche che andrò a spiegare sono valide per il 68000 ma le stesse sono anche inutili in microprocessori quali il 68040 o il 68060. La prima cosa che bisogna avere disponibile è una tavola dei cicli macchina di ogni istruzione del 68000, che troverete sintetizzata in questa lezione: dando un rapido sguardo a questa tavola vi potreste meravigliare osservando il "tempo" che impiega ogni istruzione ad essere eseguita, e forse fino a questo punto credevate che ogni istruzione fosse eseguita nello stesso tempo; ebbene vi sbagliavate!!! Infatti, come primo approccio, notate il tempo che impiega un'istruzione di moltiplicazione (`MULU`) rispetto ad una di addizione (`ADD`), e capirete immediatamente per quale motivo l'ottimizzazione è importante:

```
ADD     ; tempo di esecuzione: dai 6 ai 12+ cicli di clock

MULS    ; tempo di esecuzione: 70+ cicli di clock
```

Quindi, si capisce facilmente come ottimizzare questa istruzione:

```
lento:          MULU.W  #2,D0   ; 70+ cicli

ottimizzato:    ADD.W   d0,d0   ; 6+ cicli
```

Vi anticipo che le moltiplicazioni e le divisioni sono le due istruzioni più lente. Vediamo un approssimativo elenco delle istruzioni ordinate dalle più veloci alle più lente: (i cicli sono nella migliore ipotesi!)

```
EXT, SWAP, NOP, MOVEQ   ; 4 cicli -> le più veloci!

TST, BTST, ADDQ, SUBQ, AND, OR, EOR     ; 4 + indirizzamento, velocine...

MOVE, ADD, SUB, CMP, LEA        ; 4+ indirizzamento, ma spesso gli
; indirizzamenti sono "pesanti" da eseguire
```

Poi abbiamo BCLR/BCHG/BSET con 8+, LSR/LSL/ASR/ASL/ROR/ROL con 6 +2n, dove n è il numero di shifts da fare, infine abbiamo:

```
MULS/MULU       ; 70+ !
DIVU            ; 140+ !!
DIVS            ; 158+ !!!
```

Occorre anche ricordare che:

```
BEQ,BNE,BRA...  ; 10
DBRA            ; 10
BSR             ; 18
JMP             ; 12
RTS             ; 16
JSR             ; 16/20
```

Quindi, attenzione a non fare troppe chiamate alle subroutine, perché ogni BSR + RTS per tornare vi mangia 18+16=34 cicli almeno! Mettete sempre nel loop principale le subroutines corte, è uno spreco perdere 34 cicli di BSR + RTS per eseguire una manciata di istruzioni!

```
1  EXAMPLE:
2  BSR.S    ROUT1
3  BSR.S    ROUT2
4  BSR.S    ROUT3
5  RTS
6
7  ROUT1:
8  MOVE.W   d0,d1
9  RTS
10 ROUT2:
11 MOVEQ    #0,d2
12 MOVEQ    #0,d3
13 RTS
14 ROUT3:
15 LEA      label1(PC),A0
16 RTS
```

Versione che salva 34*3= 96 cicli:

```
1  EXAMPLEFIX:
2  MOVE.W   d0,d1
3  MOVEQ    #0,d2
4  MOVEQ    #0,d3
5  LEA      label1(PC),A0
6  RTS
```

Oltre all'istruzione in sé, conta anche il modo di indirizzamento usato. Per esempio:

```
1  MOVE.L   (a0),d0
```

è più lento di:

```
1  MOVE.L   $12(a0,d1.w),LABEL1
```

Eppure si tratta sempre di istruzioni MOVE. Comunque può apparirvi molto logico il perché della maggior lentezza della seconda istruzione rispetto alla prima: il processore deve calcolare l'offset sommando ad A0 il valore di D1 più il $12, poi fare la copia, e dove? In memoria, ad una label, anziché in un registro, cosa ben più lenta dato che i registri sono DENTRO il processore, mentre la memoria è fuori, e per raggiungerla il dato deve passare dai fili della scheda madre!!!!!

13.1 Ottimizzazioni di primo livello: lo “scambio” e la “scelta” di istruzioni

Ecco i modi di indirizzamento ordinati dal più veloce al più lento: NOTA: i numeri dopo il ; sono i cicli di clock da aggiungere al tempo usato dall'istruzione, nei casi di byte-word/longword

```
Registro dati diretto                                   Dn/An      ; 0

Registro indirizzi indiretto (o con Post-Incremento)    (An)/(An)+ ; 4/8
Immediato                                               #x         ; 4/8

Registro indirizzi indiretto con Pre-Decremento         -(An)      ; 6/10

Registro indirizzi indiretto con Offset (max 32767)     w(An)      ; 8/12
Assoluto corto                                            w        ; 8/12
Program Counter con Offset (calcolato dall'asmone)      w(PC)      ; 8/12

Program Counter con Offset e Indice                    b(PC,Rx)    ; 10/14
Registro indirizzi indiretto con Offset e Indice       b(An,Rx)    ; 10/14

Assoluto lungo                                            l        ; 12/16
```

Come si può notare, mentre un `MOVE.L LABEL1,LABEL2` solo di indirizzamento porta via 16+16 = 32 cicli, un `MOVE.L #1234,d0` porta via solo 8+0 = 8 cicli. Appare evidente come le istruzioni .W siano più veloci di quelle .L, per esempio l'indirizzamento (An), se .W impiega 4 cicli, se .L 8 cicli!

Comunque questi esempi sono **molto** indicativi, infatti anche studiando con le tabelle alla mano è difficile calcolare veramente il tempo di esecuzione della routine. Ma saremo sempre sicuri che il `BSR` è più veloce del `JSR`, che l'`ADDQ` è più veloce dell'`ADD`, e soprattutto che ogni volta che si riesce a sostituire un `MULU/DIVU/MULS/DIVS` con qualcos'altro abbiamo certamente velocizzato il tutto!

Qua stiamo parlando di “cambi di istruzione”, ossia di piccole modifiche fatte sostituendo istruzioni lente con altre più veloci. Però l'arte delle ottimizzazioni, vera regina della scena demo, comporta anche l'utilizzazione di una tabella “precalcolata” invece di implementare una mega funzione che dà gli stessi risultati, e altre infinite cose.

Però c'è anche il rovescio della medaglia: il codice megaottimizzato con tabelle e altri stratagemmi spesso diviene meno leggibile e capibile, e meno “modificabile”. Quindi, state attenti ad evitare l'errore nel quale molti di noi sono caduti, ossia nel voler ottimizzare la routine prima di averla finita, passo passo, ad ogni costo. Questo non fa che rallentare lo sviluppo della routine in questione, specialmente se si è alle prime armi, infatti a che serve una routine megaottimizzata che calcola la prospettiva, se non riusciamo più a scriverci “intorno” la routine di disegno e rotazione del solido? O addirittura non capiamo più come mai sta funzionando?

> *Mai passare sotto ottimizzazione una routine che non è completamente terminata e funzionante; inoltre, una volta che è pronta per l'ottimizzazione, ricordarsi di tenere le copie dei listati dei vari passaggi dell'ottimizzazione, in quanto spesso occorre “tornare indietro” e modificare qualcosa!!! Poi riottimizzeremo la versione modificata!*

Questo avvertimento vi suonerà strano, perché sembra che un listato, una volta ottimizzato, diventi irriconoscibile e incomprensibile anche per l'autore. Ebbene, se è *molto* ottimizzato, questo può succedere! Comunque ricordatevi che le ottimizzazioni vanno effettuate in parti del listato che effettivamente richiedono molto tempo per essere eseguite: ad esempio, è inutile stare ad ottimizzare una routine che viene eseguita una sola volta alla startup, o una sola volta per

fotogramma. Le prime routines da ottimizzare sono quelle che vengono eseguite molte volte per fotogramma, ossia quelle nei loop DBRA, o comunque in cicli vari. Ad esempio, vediamo questo listatino:

```
1  Bau:
2  cmp.w   #$ff,$dff006     ; Aspetta il Wblank
3  bne.s   Bau
4  bsr.s   routine1
5  bsr.s   routine2
6  btst    #6,$bfe001       ; Aspetta il mouse
7  bne.s   Bau
8  rts
9
10 Routine1:
11 move.w  #label2,d6
12 move.w  d0,d1
13 move.w  d2,d3
14 and.w   d4,d5
15 rts
16
17 Routine2:
18 move.w  #200,d7
19 lea     label2(PC),a0
20 lea     label3(PC),a1
21 loop1:
22 move.w  (a0)+,d0
23 move.w  (a0)+,d1
24 add.w   d0,d5
25 add.w   d0,d6
26 move.w  d5,(a1)+
27 move.w  d5,(a2)+
28 dbra    d7,loop1
29 rts
```

In questo caso, appare evidente che il 99% del tempo lo si perde eseguendo 200 volte il loop della routine2. Di conseguenza, se si ottimizzasse questo loop rendendolo veloce il doppio, l'intero programma girerebbe al doppio della velocità, mentre se si facesse andare anche al triplo o al quadruplo della velocità la routine2, non si noterebbe nemmeno la differenza!!!!! Per vedere quante "linee raster" occupa una routine, basta usare il vecchio sistema di cambiare colore all'inizio della routine, e cambiarlo nuovamente alla sua fine. In questo modo la "striscia" del colore cambiato indicherà il tempo in "linee video" usato per l'esecuzione:

```
1  Bau:
2  cmp.w   #$90,$dff006     ; Aspetta il Wblank
3  bne.s   Bau
4  bsr.s   routine1
5  move.w  #$F00,$dff180    ; Color0: ROSSO
6  bsr.s   routine2
7  move.w  #$000,$dff180    ; Color0: NERO
8  btst    #6,$bfe001       ; Aspetta il mouse
9  bne.s   Bau
10 rts
```

In questo caso, aspettiamo la linea $90, verso la metà dello schermo, facciamo eseguire la routine1, poco importante, poi cambiamo colore (rosso), facciamo eseguire la routine2, e ricambiamo colore (nero). Apparirà a video una strisciata rossa... quello è il "tempo" in cui viene eseguita routine2. Per vedere se si migliora o si peggiora la velocità, basterà vedere se la strisciata si allunga o si accorcia. Alcuni maniaci (come il mio amico hedgehog), appiccicano un pezzo di nastro adesivo sul monitor all'altezza dell'ultima linea colorata, in modo da notare ad ogni modifica ogni lieve miglioramento o peggioramento. Io, personalmente, ci metto un dito o vado ad occhio... fate voi! Abbiamo comunque già visto questo sistema nella lezione sul blitter e in `Lezione11n1.s` e seguenti, per "visualizzare" il tempo aspettato tramite i chip CIAA/CIAB. A proposito, potreste usare anche i timer per calcolare i tempi "numericamente", ma il sistema del cambio di colore è più immediato.

Ma prima cominciamo con le ottimizzazioni elementari, che occorrerebbe saper fare “in diretta” mentre si scrive. La cosa più semplice è sapere quale istruzione scegliere tra quelle possibili, quando si vuol fare un dato compito. Infatti la stessa operazione si può fare in più modi! Ad esempio, vediamo questo listato:

```
lea     LABEL1,a0
move.l  0(a0),d0
move.l  2(a0),d1
ADD.W   #5,d0
SUB.W   #5,d1
MULU.W  #2,d0
MOVE.L  #30,d2
RTS
```

La stessa cosa, si può fare scegliendo queste istruzioni:

```
lea     LABEL1(PC),a0   ; Indirizzamento (PC) più veloce
move.l  (a0),d0         ; Non occorreva l'offset 0!!
move.l  2(a0),d1        ; Questa si lascia così
ADDQ.W  #5,d0           ; numero minore di 8, si può usare ADDQ!
SUBQ.W  #5,d1           ; idem, per SUBQ!
ADD.W   d0,d0           ; salvo 60 cicli!! D0*2 è uguale a D0+D0!!!
MOVEQ   #30,d2          ; numero minore di 127, posso usare MOVEQ!
RTS
```

La routine è mooolto più veloce, ed è sempre leggibilissima. Quindi, la prima cosa da imparare è stare attenti ad usare le istruzioni Quick dedicate come `ADDQ/SUBQ/MOVEQ` nel caso il numero sia piccolo abbastanza, a togliere le moltiplicazioni e le divisioni quando possibile, ad usare indirizzamenti relativi al (PC) o a registri+offset, anziché a LABEL nude e crude, eccetera. Con un pò di esperienza, vi verrà naturale scegliere le istruzioni più veloci, e scriverete al primo colpo listati come il secondo, anziché listati come il primo presentato, che spero già ora non scriviate!!!! Ecco un altro esempio di ottimizzazione “a scambio” di istruzioni:

```
Move.l  #3,d0           ; 12 cicli
Clr.l   d0              ; 6 cicli
Add.l   #3,a0           ; 16 cicli
;
Move.l  #5,Label        ; 28 cicli
```

Versione ottimizzata “a scambio”:

```
Moveq   #3,d0           ; 4 cicli
Moveq   #0,d0           ; 4 cicli
Addq.w  #3,a0           ; 4 cicli
;
Moveq   #5,d0           ; 4 cicli
Move.l  d0,Label        ; 20 cicli, totale 24 cicli
```

Potrei continuare ancora per molto con esempietti del genere, ma non dovete conoscere a memoria tutti i casi possibili, naturalmente! Occorre piuttosto capire “il metodo”, la filosofia del coding ottimizzato. Esistono, ad esempio, tecniche per velocizzare il caricamento di valori a 32 bit nei registri:

```
move.l  #$100000,d0     ; 12 cicli
```

Versione ottimizzata:

```
moveq   #10,d0          ; 4 cicli
Swap    d0              ; 4 cicli, totale 8 cicli
```

Altra cosa **importantissima** è che l’accesso alla memoria (ossia alle label) è molto più LENTO dell’accesso ai registri dati ed indirizzi. Quindi, è una buona abitudine il tendere ad usare tutti i registri e badare di toccare il meno possibile le label. Ad esempio il listato:

```
MOVE.L  #200,LABEL1
MOVE.L  #10,LABEL2
ADD.L   LABEL1,LABEL2
```

Si può ottimizzare MOLTISSIMO scrivendo:

```
move.l  #200,d0
moveq   #10,d1
add.l   d0,d1
```

Non fate caso alla stupidità dell'esempio, ma al fatto che mentre nel primo abbiamo fatto 4 accessi alla lentissima RAM, facendo passare i dati per i fili aggrovigliati della scheda madre, nel secondo caso tutto si è svolto all'interno della CPU, turbizzando il tutto. Se finite i registri dati, usate anche i registri indirizzi per tenere dati, piuttosto che accedere a label! Inoltre, se è possibile, usate istruzioni .W anzichè .L, per esempio il listato di prima si potrebbe riottimizzare in:

```
move.w  #200,d1
moveq   #10,d0
add.w   d0,d1
```

In questo caso le istruzioni occupano 8 cicli anzichè 12...e non è poco! State però attenti che la word alta sia azzerata e/o non serva mai!!

Comunque, le ottimizzazioni "a scambio" più proficue sono quelle che eliminano istruzioni di moltiplicazione (70 cicli) e di divisione (158 cicli), e si può dire che è nata una scienza in proposito. Il caso più semplice è quando dobbiamo dividere o moltiplicare per numeri che sono potenze di 2, perché possiamo adoperare le istruzioni di shift che impiegano come cicli macchina esattamente:

```
Lsl.w   6+2n            ; n = numero di shifts
Asr.w   6+2n
Lsr.l   8+2n
Asr.l   8+2n
```

Qui n sta ad indicare il numero di bit, ed il numero dei cicli è riferito a quando si adoperano i registri. La regola da seguire in genere è la seguente: (per `MULS` o `MULU`)

Nota: alle volte ci vuole un `EXT.L D0` prima degli `ASL` che sostituiscono i `MULS`, mentre prima di quelli che sostituiscono i `MULU` può servire una pulizia della word alta con `swap d0`, `clr.w d0`, `swap d0`.

```
MULS.w  #2,d0           | ADD.L d0,d0 ; mi pare chiaro!

MULS.w  #4,d0           | ADD.L d0,d0 ; anche questo!
| ADD.L d0,d0

MULS.w  #8,d0           | ASL.l #3,d0 ; da 8 a 256 conviene l'asl
MULS.w  #16,d0          | ASL.l #4,d0
MULS.w  #32,d0          | ASL.l #5,d0
MULS.w  #64,d0          | ASL.l #6,d0
MULS.w  #128,d0         | ASL.l #7,d0
MULS.w  #256,d0         | ASL.l #8,d0
```

Se ci sono problemi coi `MULU`, si potrebbe pulire la word alta:

```
mulu.w #n,dx -> swap dx         ;n is 2^m, 2..2^8
clr.w dx        ;(2,4,8,16,32,64,128,256)
swap dx
asl.l #m,dx
```

Per il `muls` può bastare mettere un `ext.l` prima dell'`asl`.

```
muls #n,dx ->   ext.l dx        ;n is 2^m, 2..2^8
asl.l #m,dx
```

Mentre per le DIVISIONI:

```
DIVS.w  #2,d0           | ASR.L #1,d0   ; attenzione: IGNORA IL RESTO!!!!!!!
DIVS.w  #4,d0           | ASR.L #2,d0
DIVS.w  #8,d0           | ASR.L #3,d0
DIVS.w  #16,d0          | ASR.L #4,d0
```

```
5  DIVS.w  #32,d0          | ASR.L #5,d0
6  DIVS.w  #64,d0          | ASR.L #6,d0
7  DIVS.w  #128,d0         | ASR.L #7,d0
8  DIVS.w  #256,d0         | ASR.L #8,d0
9  DIVU.w  #2,d0           | LSR.L #1,d0   ; attenzione: IGNORA IL RESTO!!!!!!!!
10 DIVU.w  #4,d0           | LSR.L #2,d0
11 DIVU.w  #8,d0           | LSR.L #3,d0
12 DIVU.w  #16,d0          | LSR.L #4,d0
13 DIVU.w  #32,d0          | LSR.L #5,d0
14 DIVU.w  #64,d0          | LSR.L #6,d0
15 DIVU.w  #128,d0         | LSR.L #7,d0
16 DIVU.w  #256,d0         | LSR.L #8,d0
```

Come sapete, dopo una divisione nella word bassa rimane il risultato e in quella alta il resto; se sostituite il `DIVS/DIVU` con uno shift invece avrete il risultato nella word bassa e la word alta azzerata... quindi **non è la stessa cosa**, state attenti! Nel caso peggiore in cui n=8 otterete un numero di cicli esattamente di 6+2*8=22 cicli per le word e 8+2*8=24 cicli per le longword, quindi il risparmio è garantito. Inoltre sappiate che su un 68020 il numero dei cicli per le istruzioni di shift è lo stesso indipendentemente dal numero di bit da spostare. Inoltre tenete anche presente l'istruzione `Swap`, che impiega 4 cicli ad essere eseguita, poiché ci può far comodo in molte situazioni in cui il numero di bit da spostare è consistente. Vediamo a questo proposito una serie di esempi:

```
1  ; Shift di 9 bit a sinistra
2
3  Lsl.l   #8,d0
4  Add.l   d0,d0
5
6  ; Shift di 16 bit a sinistra
7
8  Swap    d0
9  Clr.w   d0
10
11 ; Shift di 24 bit a sinistra
12
13 Swap    d0
14 Clr.w   d0
15 Lsl.l   #8,d0
16
17 ; Shift di 16 bit a destra
18
19 Clr.w   d0
20 Swap    d0
21
22 ; Shift di 24 bit a destra
23
24 Clr.w   d0
25 Swap    d0
26 Lsr.l   #8,d0
```

Come potete vedere le tecniche per shiftare non mancano e se ne possono ricavare moltissime, come sempre spetta a voi entrare nell'ottica giusta e cercare di fare l'ottimizzazione che cercate. Quindi per potenze di 2 non avete grossi problemi a moltiplicare e a dividere in un tempo decente. I problemi potrebbero sorgere nel caso in cui il numero non sia una potenza di due; in effetti questo è vero, ma per molti valori possiamo ancora aggirare il problema. Infatti consideriamo il caso in cui dobbiamo moltiplicare il valore, contenuto in un registro, per 3: ebbene, pensate al fatto che dovete eseguire una espressione del genere 3*x, che potete anche scrivere 2*x+x. A questo punto avete risolto il vostro problema perché il vostro codice sarà:

```
1  Move.l  d0,d1
2  Add.l    d0,d0 ; d0=d0*2
3  Add.l    d1,d0 ; d0=(d0*2)+d0
```

Consideriamo un altro caso ad esempio per n=5, allora abbiamo 5*x, ossia 4*x+x: come codice avremo che:

```
1 Move.l  d0,d1
2 Asl.l   #2,d0 ; d0=d0*4
3 Add.l   d1,d0 ; d0=(d0*4)+d0
```

Consideriamo infine un altro caso in cui n=20, allora abbiamo 20*x, ma 20*x = 4*(5*x) = 4*(4*x+x)

```
1 Move.l  d0,d1
2 Asl.l   #2,d0 ;d0=d0*4
3 Add.l   d1,d0 ;d0=(d0*4)+d0
4 Asl.l   #2,d0 ;d0=4*((d0*4)+d0)
```

In breve possiamo tentare di fare una cosa del genere, se scomponendo il numero in fattori primi notiamo che ci sono molti 2; ma fatevi sempre un conticino sul numero dei cicli per vedere se vi conviene oppure no. Molti di voi potrebbero meravigliarsi nel vedere qui trattato il modo per poter ottimizzare una semplice `MULU` o `DIVU`, ma pensate i casi in cui queste si trovino in cicli, in questo caso queste tecniche sono realmente molto utili, comunque anche se la `MULU` non si trova in un ciclo, che vi costa sostituirla con qualcosa di migliore ? Poiché siamo in argomento parliamo molto brevemente delle implementazioni delle espressioni in Assembly. Quello che vi dirò non è niente di particolare ma spesso non si presta attenzione ad un fatto banale. Quando dobbiamo implementare una funzione, di solito quello che facciamo è di caricare i valori nei registri ed effettuare tutte le operazioni. In generale, per risparmiare tempo macchina nella valutazione della funzione, conviene utilizzare i metodi di raccoglimento che si imparano alle superiori, infatti consideriamo una espressione banale:

```
a*d0+b+d1+a*d3+b*d5
```

può essere scritta come:

```
a*(d0+d3)+b*(d1+d5)
```

In questo modo risparmiamo due moltiplicazioni.

Dato che per saper scegliere la giusta istruzione basta conoscere in ogni coppia di istruzioni equivalenti quale è la più veloce, vi presento una tabella simile a quella alla fine di `68000-2.txt`, con istruzioni “lente”, ed equivalenti “veloci” da usare:

```
ISTRUZIONE esempio      | EQUIVALENTE, MA PIù VELOCE
------------------------|-----------------------------------------------
add.X #6,XXX            | addq.X #6,XXX         (massimo 8)
sub.X #7,XXX            | subq.X #7,XXX         (massimo 8)
MOVE.X LABEL,XX         | MOVE.X LABEL(PC),XX   (se nella stessa SECTION)
LEA LABEL,AX            | LEA LABEL(PC),AX      (se nella stessa SECTION)
MOVE.L #30,d1           | moveq #30,d1          (min #-128, max #+127)
CLR.L d4                | MOVEQ #0,d4           (solo per i registri dati)
ADD.X/SUB.X #12000,a3   | LEA (+/-)12000(a3),A3 (min -32768, max 32767)
MOVE.X #0,XXX           | CLR.X XXX             ; muovere #0 è stupido!
CMP.X  #0,XXX           | TST.X XXX             ; il TST dove lo lasci?
Per azzerare un reg. Ax | SUB.L A0,A0           ; meglio di "LEA 0,a0".
JMP/JSR XXX             | BRA/BSR XXX           (Se XXX è vicino)
MOVE.X #label,AX        | LEA label,AX          (solo registri indirizzi!)
MOVE.L 0(a0),d0         | MOVE.L (a0),d0        (togli l'offset se è 0!!!)
LEA     (A0),A0         | HAHAHAHA!             ; Toglila, non ha effetti!!
LEA     4(A0),A0        | ADDQ.W #4,A0          ; fino ad 8
addq.l #3,a0            | addq.w #3,a0          ; Solo reg. indirizzi, max 8
Bcc.W label             | Bcc.S label           ; Beq,Bne,Bsr... dist. <128
```

Per le moltiplicazioni e divisioni di multipli di 2 convertite in `ASL` / `ASR` vedete la tabella sopra. Seguono dei casi particolari per cambiare `MULS` / `MULU` in qualcos’altro. NOTA: Se si tratta di un

MULS, spesso occorre aggiungere un `ext.l dx` come prima istruzione, per estendere il segno a longword.

```
mul*.w #3,dx -> move.l dx,ds
add.l dx,dx
add.l ds,dx
-----------------------------------
mul*.w #5,dx -> move.l dx,ds
asl.l #2,dx
add.l ds,dx
-----------------------------------
mul*.w #6,dx -> add.l dx,dx
move.l dx,ds
add.l dx,dx
add.l ds,dx
-----------------------------------
mul*.w #7,dx -> move.l dx,ds
asl.l #3,dx
sub.l ds,dx
-----------------------------------
mul*.w #9,dx -> move.l dx,ds
asl.l #3,dx
add.l ds,dx
-----------------------------------
mul*.w #10,dx -> add.l dx,dx
move.l dx,ds
asl.l #2,dx
add.l ds,dx
-----------------------------------
mul*.w #12,dx -> asl.l #2,dx
move.l dx,ds
add.l dx,dx
add.l ds,dx
-----------------------------------
mulu.w #12,dx -> swap dx        ; HEI! spesso occorre azzerare la word
clr.w dx                        ; alta per i MULU... considerate questo anche
swap dx ; per i mulu #3, #5, #6....

asl.l #2,dx                     ; normale mulu #12
move.l dx,ds
add.l dx,dx
add.l ds,dx
-----------------------------------
```

Se dovete azzerare la word alta dei registri molte volte, potete usare anche:

```
1  move.l  #$0000FFFF,ds   ; serve 1 registro per tenere $FFFF
2
3  and.l   ds,dx           ; questo è più veloce dello swappaggio, ma
4  ; richiede un registro che contenga $0000FFFF,
5  ; altrimenti "AND.L #$FFFF,dx" non è più
6  ; veloce...
```

In sintesi, ricordatevi che in caso di MULS, dato che è SIGNED, può essere necessario eseguire un EXT.L all’inizio. Invece, nel caso dei MULU, può essere necessario azzerare la word alta del registro.

Ora degli scambi “composti”:

```
asl.x #2,dy -> add.x dy,dy
add.x dy,dy
-----------------------------------
asl.l #16,dx -> swap dx
```

```
clr.w dx
-----------------------------------
asl.w #2,dy -> add.w dy,dy
add.w dy,dy
-----------------------------------
asl.x #1,dy -> add.x dy,dy
-----------------------------------
asr.l #16,dx -> swap dx
ext.l dx
-----------------------------------
bsr label -> bra label
rts
-----------------------------------
clr.x n(ax,rx) -> move.x ds,n(ax,rx)    ; ds deve essere 0, naturalmente!
-----------------------------------
lsl.l #16,dx -> swap dx
clr.w dx
-----------------------------------
move.b #-1,(ax) -> st (ax)
-----------------------------------
move.b #-1,dest -> st dest
-----------------------------------
move.b #x,mn   -> move.w #xy,mn
move.b #y,mn+1
-----------------------------------
move.x ax,ay -> lea n(ax),ay            ; -32767 <= n <= 32767
add.x #n,ay
-----------------------------------
move.x ax,az -> lea n(ax,ay),az         ;  az=n+ax+ay, n<=32767
add.x #n,az
add.x ay,az
-----------------------------------
sub.x #n,ax -> lea -n(ax),ax            ; -32767 <= n <= -9, 9 <= n <= 32767
-----------------------------------
```

A questo punto vedetevi il tempo di esecuzione delle varie istruzioni. Al tempo di esecuzione dell'istruzione, va aggiunto il tempo speso per i vari indirizzamenti, il cui tempo di esecuzione è stato visto prima. State attenti al fatto che si tratta dei tempi di esecuzione del normale 68000! Per esempio, nel 68040 i `MULS`/`MULU` sono implementati via hardware e prendono pochi cicli!

>>> MOVE.B e MOVE.W <<<

	DESTINAZIONE								
SORG.	Dn	An	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn)*	(xxx.W)	(xxx).L
Dn / An	4	4	8	8	8	12	14	12	16
(An)	8	8	12	12	12	16	18	16	20
(An)+	8	8	12	12	12	16	18	16	20
-(An)	10	10	14	14	14	18	20	18	22
(d16,An)	12	12	16	16	16	20	22	20	24
(d8,An,Xn)*	14	14	18	18	18	22	24	22	26
(xxx).W	12	12	16	16	16	20	22	20	24
(xxx).L	16	16	20	20	20	24	26	24	28
(d16,PC)	12	12	16	16	16	20	22	20	24
(d8,PC,Xn)*	14	14	18	18	18	22	24	22	26
#(data)	8	8	12	12	12	16	18	16	20

```
+-------------+----+----+----+-----+-----+--------+-----------+-------+-------+
* La grandezza del registro indice (Xn) (.w o .l) non cambia la velocità.


>>>                                MOVE.L                          <<<

+-------------+-------------------------------------------------------------+
|             |                          DESTINAZIONE                       |
+   SORG.     +-------------------------------------------------------------+
|             | Dn | An |(An)|(An)+|-(An)|(d16,An)|(d8,An,Xn)*|(xxx.W)|(xxx).L|
+-------------+----+----+----+-----+-----+--------+-----------+-------+-------+
| Dn o  An    | 4  | 4  | 12 | 12  | 12  |   16   |    18     |  16   |  20   |
| (An)        | 12 | 12 | 20 | 20  | 20  |   24   |    26     |  24   |  28   |
+-------------+----+----+----+-----+-----+--------+-----------+-------+-------+
| (An)+       | 12 | 12 | 20 | 20  | 20  |   24   |    26     |  24   |  28   |
| -(An)       | 14 | 14 | 22 | 22  | 22  |   26   |    28     |  26   |  30   |
| (d16,An)    | 16 | 16 | 24 | 24  | 24  |   28   |    30     |  28   |  32   |
+-------------+----+----+----+-----+-----+--------+-----------+-------+-------+
| (d8,An,Xn)* | 18 | 18 | 26 | 26  | 26  |   30   |    32     |  30   |  34   |
| (xxx).W     | 16 | 16 | 24 | 24  | 24  |   28   |    30     |  28   |  32   |
| (xxx).L     | 20 | 20 | 28 | 28  | 28  |   22   |    34     |  32   |  36   |
+-------------+----+----+----+-----+-----+--------+-----------+-------+-------+
| (d,PC)      | 16 | 16 | 24 | 24  | 24  |   28   |    30     |  28   |  32   |
| (d,PC,Xn)*  | 18 | 18 | 26 | 26  | 26  |   30   |    32     |  30   |  34   |
| #(data)     | 12 | 12 | 20 | 20  | 20  |   24   |    26     |  24   |  28   |
+-------------+----+----+----+-----+-----+--------+-----------+-------+-------+
* La grandezza del registro indice (Xn) (.w o .l) non cambia la velocità.
```

Ed ora le altre istruzioni.

```
Note:

#  - Operando Immediato
An - Registro indirizzi
Dn - Registro Dati
ea - Un operando specificato da un Indirizzo effettivo
M  - Indirizzo effettivo
+  - Aggiungere il tempo speso a calcolare l'indirizzo (indirizzmento)

+-------------+-----------+------------+-----------+-----------+
| Istruzione  |   Size    | op<ea>,An 1| op<ea>,Dn | op Dn,<M> |
+-------------+-----------+------------+-----------+-----------+
|             | Byte,Word |     8+     |     4+    |     8+    |
|  ADD/ADDA   +-----------+------------+-----------+-----------+
|             |   Long    |     6+     |     6+    |    12+    |
+-------------+-----------+------------+-----------+-----------+
|             | Byte,Word |     -      |     4+    |     8+    |
|  AND        +-----------+------------+-----------+-----------+
|             |   Long    |     -      |     6+    |    12+    |
+-------------+-----------+------------+-----------+-----------+
|             | Byte,Word |     6+     |     4+    |     -     |
|  CMP/CMPA   +-----------+------------+-----------+-----------+
|             |   Long    |     6+     |     6+    |     -     |
+-------------+-----------+------------+-----------+-----------+
|  DIVS       |     -     |     -      |   158+    |     -     |
+-------------+-----------+------------+-----------+-----------+
|  DIVU       |     -     |     -      |   140+    |     -     |
+-------------+-----------+------------+-----------+-----------+
|             | Byte,Word |     -      |     4     |     8+    |
|  EOR        +-----------+------------+-----------+-----------+
|             |   Long    |     -      |     8     |    12+    |
+-------------+-----------+------------+-----------+-----------+
```

MULS/MULU	-	-	70+	-
OR	Byte,Word	-	4+	8+
	Long	-	6+	12+
SUB	Byte,Word	8+	4+	8+
	Long	6+	6+	12+

Istruzione	Size	op #,Dn	op #,An	op #,M
ADDI	Byte,Word	8	-	12+
	Long	16	-	20+
ADDQ	Byte,Word	4	4	8+
	Long	8	8	12+
ANDI	Byte,Word	8	-	12+
	Long	14	-	20+
CMPI	Byte,Word	8	-	8+
	Long	14	-	12+
EORI/SUBI	Byte,Word	8	-	12+
	Long	16	-	20+
MOVEQ	Long	4	-	-
ORI	Byte,Word	8	-	12+
	Long	16	-	20+
SUBQ	Byte,Word	4	8	8+
	Long	8	8	12+

Istruzione	Size	Register	Memory
NBCD	Byte	6	8+
CLR/NEG	Byte,Word	4	8+
NEGX/NOT	Long	6	12+
Scc	Byte,False	4	8+
	Byte,True	6	8+
TAS	Byte	4	14+
TST	Byte,Word,Long	4	4+

LSR/LSL	Byte,Word	6 + 2n	8+
ASR/ASL			
ROR/ROL	Long	8 + 2n	-
ROXR/ROXL			

nota: n è il numero di shifts!

Bit Manipulation Istruzione Execution Times

		Dynamic		Static	
Istruzione	Size	Register	Memory	Register	Memory
BCHG/BSET	Byte	-	8+	-	12+
	Long	8	-	12	-
BCLR	Byte	-	8+	-	12+
	Long	10	-	14	-
BTST	Byte	-	4+	-	8+
	Long	6	-	10	-

Istruzione	Displacement	Branch Taken	Branch Not Taken
Bcc	Byte	10	8
	Word	10	12
BRA	Byte	10	-
	Word	10	-
BSR	Byte,word	18	-
DBcc	cc true	-	12
	cc false, Count Not Expired	10	_
	cc false, Counter Expired	_	14

Ins.	Sz	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn)+	(x).W	(x).L	(d16,PC)	(d8,PC,Xn)*
JMP	-	8	-	-	10	14	10	12	10	14
JSR	-	16	-	-	18	22	18	20	18	22
LEA	-	4	-	-	8	12	8	12	8	12
PEA	-	12	-	-	16	20	16	20	16	20
	Word	12+4n	12+4n	_	16+4n	18+4n	16+4n	20+4n	16+4n	18+4n

```
|MOVEM+----+-----+-----+----+-----+-----------+-----+-----+--------+----------+
|M->R |Long|12+8n|12+8n| _  |16+8n|  18+8n    |16+8n|20+8n| 16+8n  |  18+8n   |
|     |    |     |     |    |     |           |     |     |        |          |
+-----+----+-----+-----+----+-----+-----------+-----+-----+--------+----------+
|     |Word| 8+4n|  _  |8+4n|12+4n|  14+4n    |12+4n|16+4n|   _    |    _     |
|     |    |     |     |    |     |           |     |     |        |          |
|MOVEM+----+-----+-----+----+-----+-----------+-----+-----+--------+----------+
|R->M |Long| 8+8n|  _  |8+8n|12+8n|  14+8n    |12+8n|16+8n|   _    |    _     |
|     |    |     |     |    |     |           |     |     |        |          |
+-----+----+-----+-----+----+-----+-----------+-----+-----+--------+----------+
nota: n è il numero di registri da muovere.


EXT/SWAP/NOP    4
EXG             6
UNLK            12
LINK/RTS        16
RTE             20
```

Considerate infine che le exceptions richiedono 44 cicli se si tratta di un interrupt, 34 se si tratta di un TRAP. Più 20 per l'RTE!!! Mi raccomando di commentare **sempre** una ottimizzazione, ad esempio, supponiamo vogliate ottimizzare questa routine:

```
1  movem.l label1(PC),d1-d4
2  mulu.w  #16,d1
3  mulu.w  #3,d2
4  muls.w  #5,d3
5  divu.w  #8,d4
6  rts
```

Ottimizzandola, il risultato sarebbe:

```
1  movem.l label1(PC),d1-d4
2  asl.l   #4,d1           ; mulu.w #16,d1
3  move.l  d2,d5           ; \
4  add.l   d2,d2           ;  > mulu.w #3,d2
5  add.l   d5,d2           ; /
6  move.l  d3,d5           ; \
7  asl.l   #2,d3           ;  > muls.w #5,d3
8  add.l   d5,d3           ; /
9  asr.l   #3,d4           ; divu.w #8,d4
10 rts
```

Oltre ad aver usato il registro d5, abbiamo reso più difficile da leggere il listato. A prima vista, se non avessimo messo i commenti, si capirebbe cosa succede ai registri d1,d2,d3 e d4? E immaginatevi se avessimo dovuto anche pulire la word alta prima dei MULU e estendere prima dei MULS:

```
1  movem.l label1(PC),d1-d4
2  swap    d1
3  clr.w   d1
4  swap    d1
5  asl.l   #4,d1
6  swap    d2
7  clr.w   d2
8  swap    d2
9  move.l  d2,d5
10 add.l   d2,d2
11 add.l   d5,d2
12 ext.l   d3
13 move.l  d3,d5
14 asl.l   #2,d3
15 add.l   d5,d3
16 asr.l   #3,d4
17 rts
```

Oppure, si può azzerare la word alta nel modo più veloce:

```
move.l  #$FFFF,d6
...
movem.l label1(PC),d1-d4
and.l   d6,d1
asl.l   #4,d1
and.l   d6,d2
move.l  d2,d5
add.l   d2,d2
add.l   d5,d2
ext.l   d3
move.l  d3,d5
asl.l   #2,d3
add.l   d5,d3
asr.l   #3,d4
rts
```

Se tornate al vostro listato, dopo 1 mese dalla scrittura, quanto ci mettereste a capire che tutte queste istruzioni incomprensibili non fanno altro che 3 moltiplicazioni e una divisione? **Ci mettereste molto**, o dovreste addirittura cancellare il listato e ripartire da capo in caso di modifica. Non ho messo i commenti a quest'ultima versione proprio per farvi capire come sia FONDAMENTALE mettere i commenti alle ottimizzazioni, come nel listato precedente. Quindi: **commentate sempre le ottimizzazioni!!!!!!!!!!!!**

Altro esempio: vedetevi queste 3 istruzioni:

```
move.l  a1,a0
add.w   #80,a0
add.l   d0,a0
```

La stessa cosa si può fare con:

```
lea     80(a1,d0.l),a0  ; oppure d0.w se basta la word bassa di d0.
```

13.2 Ottimizzazioni di secondo livello: il “tabellamento”

Parliamo ora delle tabelle, un argomento tra i più importanti per l'Ottimizzazione, quella con la O maiuscola, che permette di andare più veloci di qualsiasi compilatore C, BASIC, eccetera. Le tabelle per l'ottimizzazione sono “simili” a quelle usate per contenere le coordinate dell'ondeggiamento degli sprite o altro, che abbiamo visto nelle lezioni precedenti: in quel caso si può dire che "precalcolavamo" le varie posizioni che avrebbero assunto gli oggetti, ma qua la tabella viene usata per “precalcolare” i risultati di una data moltiplicazione, divisione, o di intere funzioni matematiche, quindi il caso è un pò diverso. Facciamo un esempio concreto. Supponiamo di avere una routine che elabora una serie di valori compresi tra 0 e 100, ed a un certo punto bisogna eseguire una moltiplicazione per una costante c. Ora, se quella routine deve essere eseguita molte volte, allora quella moltiplicazione ci farà perdere un bel pò di tempo. Come aggirare il problema? Ci creiamo una tabella contenete tutti i valori del nostro “range” (serie) da 0...100 già moltiplicati per c, cioè una cosa del genere:

```
Table:
dc.w    0*c
dc.w    1*c
dc.w    2*c
dc.w    3*c
.
dc.w    n*c
.
dc.w    100*c
```

A questo punto è facile accedere alla tabella, perché dato il valore da moltiplicare per c in d0, avremo che:

```
Lea     Table,a0        ; Indirizzo della tabella
Add.w   d0,d0           ; d0 * 2, per trovare l'offset nella tabella,
```

```
3  ; dato che ogni suo valore è lungo 1 word.
4  Move.w  (a0,d0.w),d0     ; Copia il valore giusto dalla tabella in d0
```

Facile, no? L'unico svantaggio è che abbiamo il listato 100 words più lungo, per contenere la tabella. Se tale tabella poi non fosse più lontana di 256 bytes, potremmo scrivere:

```
1  Add.w   d0,d0                ; d0*2, ogni val. 1 word, ossia 2 byte
2  Move.w  Table(pc,d0.w),d0    ; copia dalla tab. il valore giusto
```

Se il listato fosse per 68020+, basterebbe una sola istruzione:

```
1  Move.w  Table(pc,d0.w*2),d0     ; istruzione del 68020 o superiori
```

Quest'ultima però è una anticipazione, infatti le ottimizzazioni specifiche per 68020 le tratteremo in seguito. Comunque, la soluzione più adottata per tabelle "corte" è quella di costruirsele in una `section BSS` tramite una routine. In questo modo, il file eseguibile non risulta più lungo, ma occupa solamente un poco più di memoria (a meno che non facciate una tabella lunga 500Kb, in tal caso occupa MOLTA più memoria, heheheeh!)

Se siete stati attenti, nelle lezioni precedenti abbiamo già "tabellato" un paio di listati: uno per togliere una `MULU.W #40`, molto frequente dato che 40 è la lunghezza di una linea di schermo lowres. Rivedetevi attentamente quell'esempio, si tratta di `Lezione8n2.s`, dove sono presenti sia la versione ottimizzata che quella normale a confronto. Rivedetevi anche i listati precedenti per vedere le routine normale e ottimizzata da sole. Il problema era un:

```
1  mulu.w  #largschermo,d1          ; Ossia mulu.w #40,d1
```

Per "risolverlo", ecco lo stratagemma:

```
1  ; PRECALCOLIAMO UNA TABELLA CON I MULTIPLI DI 40, ossia della larghezza dello
2  ; schermo, per evitare di fare una moltiplicazione per ogni plottaggio.
3
4  —.—     —.—     —.—     —.—     —.—     —.—     —.—     —.—     —.—     —.—
5
6  lea     MulTab,a0        ; Indirizzo spazio di 256 words dove scrivere
7  ; i multipli di 40...
8  moveq   #0,d0            ; Iniziamo da 0...
9  move.w  #256—1,d7        ; Numero di multipli di 40 necessari
10 PreCalcLoop
11 move.w  d0,(a0)+         ; Salviamo il multiplo attuale
12 add.w   #LargSchermo,d0  ; aggiungiamo larghschermo, prossimo multiplo
13 dbra    d7,PreCalcLoop   ; Creiamo tutta la MulTab
14 ....
15
16 SECTION Precalc,bss
17
18 MulTab:
19 ds.w    256     ; notare che la section bss, composta da zeri, non
20 ; allunga la lunghezza effettiva del file eseguibile.
```

Questo per quanto riguarda il calcolo della tabella. Poi, al posto del `MULU`:

```
1  lea     MulTab,a1        ; Indirizzo della tabella con i multipli della
2  ; largh. schermo precalcolati in a1
3  add.w   d1,d1            ; d1*2, per trovare l'offset nella tabella
4  add.w   (a1,d1.w),d0     ; copia il multiplo giusto da tab a d0
```

Questo, in breve, è il metodo per tabellare una moltiplicazione. Naturalmente, qua sapevamo che d1 poteva andare solo da 0 a 255, di conseguenza abbiamo precalcolato solo 256 multipli. Se invece d1 avesse avuto un range da 0 a 65000, avremmo dovuto fare una tabella lunga 128Kb, e ciò potrebbe anche non convenire! Se il risultato massimo nella tabella non supera $FFFF, ossia 65535, basta fare una tabella con valori .Word. Se invece i valori più alti superano tale valore, la tabella deve essere fatta di longword. In questo caso, dovremo cambiare il modo di trovare l'offset: non più *2, ma *4!

```
lea     MulTab,a1       ; Indirizzo della tabella con i multipli della
; largh. schermo precalcolati in a1
add.w   d1,d1           ; d1*4, per trovare l'offset nella tabella
add.w   d1,d1           ;
move.l  (a1,d1.w),d0    ; copia il multiplo giusto da tab a d0
```

Per quanto riguarda il tabellamento delle divisioni, la cosa è analoga, basta farsi una routine con un loop che divide ogni ciclo per un numero crescente e salvare i risultati nella tabella. In questo caso si può scegliere di salvare solo la word bassa, con il risultato, o anche quella alta con il resto, se serve al nostro scopo.

Una cosa fondamentale è di crearsi “in loco” la tabella, **mai inserire una tabella, specialmente se lunga diversi KB, già calcolata**. Per esempio, se precalcolassimo una multab da 20KB, immaginatevi la differenza tra un eseguibile che la calcola alla startup, e una che la ha inclusa con incbin o include già precalcolata: (esempio)

```
file1   ->      lunghezza = 40K         ; calcola la tab all'inizio
file1   ->      lunghezza = 60K         ; ha la tab inclusa con incbin
```

A livello di consumo della memoria sono in parità, ma se doveste fare una 40K intro o una 64K intro immaginatevi l'immenso risparmio di spazio, a scapito di 1 secondo o 2 di precalcolamento all'inizio. Ma anche se aveste fatto un gioco o un programma, il fatto di essere più costo di 20k (o più) vi permetterebbe di inserire più cose nel disco e una maggior diffusione nelle bbs data la sua brevità. Poi c'è ancora un incentivo a precalcolare sul posto le tabelle: il fatto che si può facilmente modificare il listato, ad esempio volendo moltiplicare per 80 anziché per 40. Il FESSO che ha incluso con l'INCBIN una tabellona di multipli di 40, dovrebbe riscriversi la routine di moltiplicazione per 80, eseguirla, salvare il file binario, mentre il FURBO che ha la routine di creazione nel listato deve cambiare semplicemente 40 in 80, e fa tutto da se. Infine, specialmente per precalcolamenti di routine complesse, appare **molto** più chiaro il funzionamento se si ha sott'occhio la routine originaria che crea la tabella. Quindi, **precalcolate sempre “sul posto” le tabelle in spazi azzerati, specialmente in SECTION BSS se sono tabelle grandi**.

Il consiglio che vi posso dare è cercare sempre di tabellare **tutto**.

Se siete stati attentissimi, dovreste ricordarvi anche che nella lezione11 un listato ha subito una ottimizzazione da tabellamento, ben più azzardata di quella vista ora. Infatti si tabella un'intera routine, anziché una sola moltiplicazione. Non a caso l'ho messa nella Lezione11 e non nella 8! Il listato “normale” è `Lezione11l5.s`, quella “tabellata” `Lezione11l5b.s`. Rivedetevi come è avvenuta la forte ottimizzazione, che ripropongo.

Questa è la routine “normale”:

```
Animloop:
moveq   #0,d0
move.b  (A0)+,d0        ; Prossimo byte in d0
MOVEQ   #8-1,D1         ; 8 bit da controllare e espandere.
BYTELOOP:
BTST.l  D1,d0           ; Testa il bit del loop attuale
BEQ.S   bitclear        ; è azzerato?
ST.B    (A1)+           ; Se no, setta il byte (=$FF)
BRA.S   bitset
bitclear:
clr.B   (A1)+           ; Se è azzerato, azzera il byte
bitset:
DBRA    D1,BYTELOOP     ; Controlla ed espandi tutti i bit del byte
DBRA    D7,Animloop     ; Converti tutto il fotogramma
```

Non si è fatto altro che precalcolare tutte le possibilità:

```
;***************************************************************************
; Routine che precalcola tutti i possibili 8 bytes abbinati ai possibili
; 8 bit. Per tutti si intende $FF, ossia 255.
;***************************************************************************

```

```
6  PrecalcoTabba:
7  lea      Precalctabba,a1 ; Destinazione
8  moveq    #0,d0           ; Parti dal valore zero
9  FaiTabba:
10 MOVEQ    #8-1,D1         ; 8 bit da controllare e espandere.
11 BYTELOOP:
12 BTST.l   D1,d0           ; Testa il bit del loop attuale
13 BEQ.S    bitclear        ; è azzerato?
14 ST.B     (A1)+           ; Se no, setta il byte (=$FF)
15 BRA.S    bitset
16 bitclear:
17 clr.B    (A1)+           ; Se è azzerato, azzera il byte
18 bitset:
19 DBRA     D1,BYTELOOP     ; Controlla ed espandi tutti i bit del byte:
20 ; D1 calando ogni volta fa fare il btst di
21 ; tutti i bit.
22 ADDQ.W   #1,D0           ; Prossimo valore
23 CMP.W    #256,d0         ; Li abbiamo fatti tutti? (max $FF)
24 bne.s    FaiTabba
25 rts
```

E cambiare la routine “esecutiva”:

```
1  Animloop:
2  moveq    #0,d0
3  move.b   (A0)+,d0        ; Prossimo byte in d0
4  lsl.w    #3,d0           ; d0*8 per trovare il valore nella tabba
5  ; (ossia l'offset dal suo inizio)
6  lea      Precalctabba,a2
7  lea      0(a2,d0.w),a2   ; In a2 l'indirizzo nella tabba degli 8 byte
8  ; giusti per "l'espansione" degli 8 bit.
9  move.l   (a2)+,(a1)+     ; 4 bytes espansi
10 move.l   (a2),(a1)+      ; 4 bytes espansi (totale 8 bytes!!)
11
12 DBRA     D7,Animloop     ; Converti tutto il fotogramma
```

Come vedete, qua stiamo entrando in un tipo di ottimizzazione che per essere fatto occorre avere una certa esperienza, e un certo intuito. Meccanicamente è facile dire: “provo a tabellare tutte le moltiplicazioni e le divisioni, e metto tutti gli `ADDQ` e i `MOVEQ` possibili”. Ma quando ci so trovano davanti routines “strane” tipo quella già vista che fa dei `BTST` di tutto un byte e lo espande ad 8 bytes, occorre avere l’occhio di lince per intuire come ottimizzarla. È questo occhio di lince che fa la differenza tra una routine 3d che va a scatti quando gira 10 punti, e una che va al cinquantesimo di secondo pur ruotandone 8192. E naturalmente non si può fare un elenco di tutte le possibili routines con accanto tutte le possibili ottimizzazioni. Occorre farsi l’occhio di lince vedendo i pochi esempi presentati.

13.3 Ottimizzazioni varie - gruppo misto

Consideriamo il caso in cui dobbiamo eseguire una determinata routine per ogni valore in d0, ed inoltre supponiamo che questi possibili valori siano compresi tra 0 e 10. Ebbene, potremmo essere tentati di fare una cosa del genere:

```
1  Cmp.b    #1,d0
2  Beq.s    Rout1
3  Cmpi.b   #2,d0
4  Beq.s    Rout2
5  ...
6  Cmp.b    #10,d0
7  Beq.s    Rout10
```

è una pessima idea, infatti come minimo avremmo potuto fare così:

```
1  Subq.b   #1,d0    ; togliamo 1. Se d0=0, allora si imposta il flag Z
2  Beq.s    Rout1    ; Di conseguenza d0 era 1, e saltiamo a Rout1
3  Subq.b   #1,d0    ; eccetera...
4  Beq.s    Rout2
```

```
...
Subq.b  #1,d0
Beq.s   Rout10
```

In effetti così è già meglio, ma noi siamo perfezionisti e con l'aiuto di una tabella facciamo così:

```
Add.w   d0,d0               ;\ d0*4, per trovare l'offset nella tabella,
Add.w   d0,d0               ;/       costituito da longwords (4 bytes!)
Move.l  Table(pc,d0.w),a0 ; In a0 l'indirizzo della routine giusta
Jmp     (a0)

Table:
dc.l    Rout1   ; 0 (valore in d0 per richiamare la routine)
dc.l    Rout2   ; 1
dc.l    Rout3   ; 2
dc.l    Rout4   ; 3
dc.l    Rout5   ; 4
dc.l    Rout6   ; 5
dc.l    Rout7   ; 6
dc.l    Rout8   ; 7
dc.l    Rout9   ; 8
dc.l    Rout10  ; 9
```

In questo modo non facciamo confronti, ed è ovvio che è un'ottima tecnica nel caso in cui conosciamo i valori da confrontare e sono consecutivi. Vorrei farvi notare inoltre che se utilizziamo in modo intensivo le tabelle potremmo addirittura lavorare con le potenze del due, risparmiandoci quindi quelle due `ADD.w`. Quindi, quando si vuole la routine 1, occorrerà d0=0, quando si vuole Rout2 d0=4, quando si vuole Rout3 d0=8, eccetera.

Ci sono anche delle varianti di questo sistema, ad esempio:

```
move.b  Table(pc,d0.w),d0       ; Prendi dalla tavola l'offset giusto
jmp     Table(pc,d0)            ; aggiungilo a Table, e salta!

Table:
dc.b    Rout1-Table     ; 0
dc.b    Rout2-Table     ; 1
dc.b    Rout3-Table     ; 2
...
even
```

Con questo sistema non dobbiamo moltiplicare d0, perché abbiamo fatto una tabella di offset delle routines dalla tabella stessa. Qua sono offset .byte, perché si suppone che le routines siano piccole e vicine. Altrimenti gli offset potranno essere .word:

```
add.w   d0,d0                   ; d0*2
move.w  Table(pc,d0.w),d0       ; Prendi dalla tavola l'offset giusto
jmp     Table(pc,d0)            ; aggiungilo a Table, e salta!

Table:
dc.w    Rout1-Table     ; 0
dc.w    Rout2-Table     ; 1
dc.w    Rout3-Table     ; 2
...
```

Il vantaggio di questo sistema è che non occorre moltiplicare per 4 il registro d0, ma solo per 2. Se non riuscite a mettere la tabella abbastanza vicina, potete fare così:

```
add.w   d0,d0                   ; d0*2
lea     Table(pc),a0
move.w  (a0,d0.w),d0
jmp     (a0,d0.w)

Table:
dc.w    Rout1-Table     ; 0
dc.w    Rout2-Table     ; 1
dc.w    Rout3-Table     ; 2
...
```

Già che abbiamo implementato il salto alle routine utilizzando `SUBQ.b #1,d0` seguito dai `BEQ`, senza né `CMP` né `TST`, occupiamoci degli usi di tale particolarità, legata ai Condition Codes

(rivedeteveli bene in `68000-2.txt`) Noi programmatori in assembly ci possiamo prendere il lusso di testare tre condizioni alla volta, infatti consideriamo l'esempio:

```
1  Add.w   #x,d0          ; i cc sono settati in qualche modo
2  Beq.s   Zero           ; il risultato è zero
3  Blt.s   Negativo       ; il risultato è minore di zero
4  ...                    ; Altrimenti il risultato è positivo...
```

Quindi, se dovete testare qualche risultato, cercate sempre di farlo dopo l'ultima operazione matematica, e non alla fine quando i cc indicheranno tutt'altra cosa. Sarebbe bene se conosceste quali cc influenzano le varie istruzioni. Inoltre vi consiglio di piazzare le `BCC` a seconda della loro probabilità di essere eseguite per prime, cioè praticamente quelle che fanno trasferire il controllo con più probabilità. Ad esempio un altro caso interessante è questo: abbiamo un certo numero di valori, non sappiamo quanti, ma sappiamo che terminano con uno zero...supponiamo che dobbiamo copiarli da una zona di memoria ad un altra. Potremmo fare una cosa del genere:

```
1  Lea     Source,a0
2  Lea     Dest,a1
3  CpLoop:
4  Move.b  (a0)+,d0       ; sorgente -> d0
5  Move.b  d0,(a1)+       ; d0 -> destinazione
6  Tst.b   d0             ; d0=0?
7  Bne.s   CpLoop         ; Se non ancora, continua
```

Ma possiamo fare di meglio nel seguente modo:

```
1  Lea     Source,a0
2  Lea     Dest,a1
3  CpLoop:
4  Move.b  (a0)+,(a1)+    ; sorgente -> destinazione
5  Bne.s   CpLoop         ; flag 0 settato? Se non ancora, continua!
```

Come potete vedere, il 68000 in questo caso fa tutto da solo.

Parliamo ora delle chiamate alle subroutine, e quindi delle `MOVEM`. L'utilizzo delle subroutine è evidentemente utilissimo nella stesura dei programmi, ma al momento di ottimizzare il vostro codice occorre notare che invece di utilizzare la coppia di istruzioni `BSR label / RTS` potreste anche utilizzare `BRA label`, seguita alla fine della soubroutine con una un'altra `BRA` che vi riporta alla istruzione immediatamente successiva a `JMP label`, ma questa ottimizzazione è a vostra discrezione. Comunque utilizzate se potete sempre `BSR` invece di `JSR` ed analogamente `BRA` invece di `JMP`, sempre che sia possibile. Ritornando comunque all'utilizzo delle routine, capita spesse volte di dover azzerare i contenuti dei registri prima di iniziare a lavora su di essi, tuttavia possiamo risparmiarci ogni volta una sfilza di `MOVEQ #0,Dx` e `SUB.l Ax,Ax`, infatti facciamo questa operazione all'inizio del programma principale e vediamo cosa accade quando chiamiamo le nostre subroutine, esempio:

```
1  Moveq   #0,d0   ;
2  Moveq   #0,d1
3  ...
4  Moveq   #0,d7
5  Move.l  d0,a0
6  ..
7  Move.l  d0,a6
8  Main:
9  Bsr.s   Pippo
10 Bsr.s   Pluto
11 Bsr.s   Paperino
12 ...
13 Bra.s   Main
```

Ebbene se salviamo ad ogni chiamata i contenuti dei registri utilizzati avremo che ogni volta che termina una routine si andrà alla prossima già con i registri "puliti", è ovvio che comunque bisogna organizzare il proprio codice per bene. Altrimenti, potreste con 1 sola istruzione pulire tutti i registri, e precisamente:

```
movem.l TantiZeri(PC),d0-d7/a0-a6

TantiZeri:
dcb.b   15,0
```

Veniamo ora all'istruzione MOVEM ed esaminiamo i suoi pregi e difetti. Osserviamo innanzitutto il numero di cicli macchina della MOVEM, soprattutto nei trasferimenti di longword: nei trasferimenti dai registri alla memoria impiega 8+8n, dove n indica il numero di registri, osserviamo inoltre il numero di cicli che invece impiega una semplice MOVE.l Dx,(Ax): 12 cicli. Il solito ingegnere si potrebbe porre allora la seguente domanda: nel caso devo trasferire più longword contenute in registri tutti differenti, fino a che punto mi conviene utilizzare la classica MOVE.l Dx,(Ax) ? Ebbene anche questa volta l'ingegnere a fatto una giusta osservazione, consideriamo infatti un caso limite in cui dobbiamo traferire il contenuto dei registri D0..D7 e A0..A6: avremmo bisogno esattamente di 8+7=15 MOVE.l per un totale di 15*12=180 cicli. Invece, se utilizziamo la MOVEM, avremmo 8+8*15=128 cicli, cioè un risparmio di 52 cicli! È evidente a questo punto che bisogna utilizzare la mastodontica MOVEM quando bisogna trasferire grosse quantità di dati, tuttavia nel caso siano coinvolti solamente due registri si può utilizzare ancora la MOVE.l normale. Vediamo a questo punto una serie di applicazioni pratiche che partono da un codice non ottimizzato fino ad arrivare ad uno ottimizzato. Supponiamo per esempio di dover azzerare 1200 byte a partire dalla locazione Table; i principianti farebbero in questo modo:

```
Lea     Table,a0        ; 12 cicli
Move.w  #1200-1,d7      ; 8 cicli
CleaLoop:
Clr.b   (a0)+           ; 8 cicli
Dbne    d7,CleaLoop
```

Questo tipo di codice è orrido!! Infatti vediamo quanto impiega...le prime due istruzioni impiegano 20 cicli, invece la CLR.b dovrà essere eseguita 1200 volte quindi 1200*8=9600 cicli, inoltre c'è da aggiungere la DBNE che dovrà essere eseguita ben 10*1199=11990 cicli più 14 alla fine, dunque ricapitolando 20+9600+11990+14=21624 cicli!!! Bè tutto ciò non merita commento. Avremmo almeno potuto fare una cosa del genere:

```
Lea     Table,a0
Move.w  #(1200/4)-1,d7  ; numero di bytes diviso 4, per il clr.L!!
Clr:
Clr.l   (a0)+           ; azzeriamo 4 bytes alla volta...
Dbra    d7,Clr          ; e facciamo 1/4 di loops.
```

Infatti, con una CLR.l, almeno cancelliamo in un colpo 4 byte, e poiché ne dobbiamo cancellare 1200, faremmo 1200/4=300 cicli, risparmiando un bel pò rispetto a prima (fatevi voi i calcoli per pietà). Per ottimizzare ancora di più, possiamo fare questo:

```
Lea     Table,a0
Move.w  #(1200/16)-1,d7 ; numero di bytes diviso 16, per il clr.L!!
Clr:
Clr.l   (a0)+           ; azzeriamo 4 bytes
Clr.l   (a0)+           ; azzeriamo 4 bytes
Clr.l   (a0)+           ; azzeriamo 4 bytes
Clr.l   (a0)+           ; azzeriamo 4 bytes
Dbra    d7,Clr          ; e facciamo 1/16 di loops.
```

Tuttavia anche questo tipo di codice può essere classificato come pessimo, proviamo ad ottimizzarlo di più, usando un registro dati:

```
Lea     Table,a0
moveq   #0,d0           ; più veloce "move.l d0" di un "CLR"!
Move.w  #(1200/32)-1,d7 ; numero di bytes diviso 32
Clr:
move.l  d0,(a0)+                ; azzeriamo 4 bytes
move.l  d0,(a0)+
move.l  d0,(a0)+
move.l  d0,(a0)+
```

```
move.l  d0,(a0)+
move.l  d0,(a0)+
move.l  d0,(a0)+
move.l  d0,(a0)+
Dbra    d7,Clr          ; e facciamo 1/32 di loops.
```

Con questa versione abbiamo aumentato l'ottimizzazione dovuta al calo dei DBRA da eseguire, e abbiamo sfruttato il fatto che usare i registri è megaveloce, anche più del CLR.

Veniamo ora ad utilizzare la MOVEM e vediamo cosa avviene:

```
movem.l TantiZeri(PC),d0-d6/a0-a6       ; azzeriamo tutti i registri
; tranne d7 e a7, naturalmente,
; che è lo stack. Potete
; azzerarli così o con
; tanti moveq #0,Dx...

; Ora abbiamo 7+7=14 registri azzerati, per un totale di 14*4=56 byte.
; Dobbiamo fare 1200byte/56byte = 21 trasferimenti, ma 21*56 = 1176 byte, e ne
; restano da fare altri 1200-1176 = 24byte che faremo a parte.

Move.l  a7,SalvaStack   ; Salviamo lo stack in una label
Lea     Table+1200,a7   ; Mettiamo in A7 (o SP, è lo stesso registro)
; l'indirizzo della fine dell'area da pulire.
Moveq   #21-1,d7        ; Numero di movem da fare (2100/56=21)
CleaLoop:
Movem.l d0-d6/a0-a6,-(a7) ; Azzeriamo "all'indietro" 56 bytes.
; Se vi ricordate, il movem in scrittura
; lavora "all'indietro", per lo stack.
Dbra    d7,CleaLoop
Movem.l d0-d5,(a7)+       ; Azzeriamo gli altimi 24 bytes
Move.l  SalvaStack(PC),a7 ; Rimettiamo a posto lo stack in SP
rts

SalvaStack:
dc.l    0
```

Facciamo un pò di conti, la MOVEM interna occuperà esattamente 8+8*14=120 cicli, dovrà essere eseguita 21 volte, quindi 21*120=2520 cicli, a cui dovremo aggiungere tutta la fase di inizializzazione e di chiusura ma non vi preoccupate non supererà mai i casi di sopra. Possiamo essere ancora più perfezionisti espandendo il codice, cioè eliminando il cicli e piazzando tante MOVEM quante ne abbiamo bisogno; non vi spaventate, l'espansione del codice è una tecnica molto utilizzata, sopratutto quando non si sà più cosa ottimizzare, vedremo in seguito una serie di esempi. Nel caso di prima, comunque, ecco cosa succederebbe:

```
Move.l  a7,SalvaStack   ; Salviamo lo stack in una label
Lea     Table+1200,a7   ; Mettiamo in A7 (o SP, è lo stesso registro)
; l'indirizzo della fine dell'area da pulire.
CleaLoop:

rept    20              ; ripeto 20 movem...
Movem.l d0-d7/a0-a6,-(a7) ; Azzeriamo "all'indietro" 60 bytes.
endr

Move.l  SalvaStack(PC),a7 ; Rimettiamo a posto lo stack in SP
rts
```

Da notare che, avendo eliminato il DBRA, possiamo usare anche il registro d7, che ci fa azzerare 4 bytes in più per ogni MOVEM. In questo modo, 1200/60 fa 20 esatto. Le demo solitamente usano questo sistema, il più veloce!

Vediamo meglio la tecnica di espansione del codice. Osservate questa routine:

```
ROUTINE2:
MOVEQ   #64-1,D0        ; 64 cicli
SLOWLOOP2:
MOVE.W  (a2),(a1)
ADDQ.w  #4,a1
ADDQ.w  #8,a2
DBRA    D0,SLOWLOOP2
```

Ed ecco la routine molto velocizzata:

```
ROUTINE2:
MOVE.W  (a2),(a1)
MOVE.W  8(a2),4(a1)
MOVE.W  8*2(a2),4*2(a1)
MOVE.W  8*3(a2),4*3(a1)
MOVE.W  8*4(a2),4*4(a1)
MOVE.W  8*5(a2),4*5(a1)
MOVE.W  8*6(a2),4*6(a1)
MOVE.W  8*7(a2),4*7(a1)
.....
MOVE.W  8*63(a2),4*63(a1)
```

Abbiamo tolto il tempo usato per il DBRA e i 2 ADDQ! Comunque, occorre dire che nei processori 68020 e superiori sono presenti le instruction cache, che velocizzano i loop lunghi meno di 256 bytes. Quindi, può succedere di ottimizzare per 68000, e rendere meno veloce su un 68020. Di conseguenza, sarebbe bene fare una mediazione di questo tipo:

```
ROUTINE2:
MOVEQ   #4-1,D0         ; solo 4 cicli (64/16)
FASTLOOP2:
MOVE.W  (a2),(a1)               ; 1
MOVE.W  8(a2),4(a1)             ; 2
MOVE.W  8*2(a2),4*2(a1)         ; 3
MOVE.W  8*3(a2),4*3(a1)         ; 4
MOVE.W  8*4(a2),4*4(a1)         ; 5
MOVE.W  8*5(a2),4*5(a1)         ; ...
MOVE.W  8*6(a2),4*6(a1)
MOVE.W  8*7(a2),4*7(a1)
MOVE.W  8*8(a2),4*8(a1)
MOVE.W  9*9(a2),4*9(a1)
MOVE.W  8*10(a2),4*10(a1)
MOVE.W  8*11(a2),4*11(a1)
MOVE.W  8*12(a2),4*12(a1)
MOVE.W  8*13(a2),4*13(a1)
MOVE.W  8*14(a2),4*14(a1)
MOVE.W  8*15(a2),4*15(a1)       ; 16
ADD.w   #4*16,a1
ADD.w   #8*16,a2
DBRA    D0,FASTLOOP2
```

Questo anche per il CLEAR con i MOVEM e le altre routine dove ripetiamo a tappeto.

Facciamo ora un paio di osservazione utili: il metodo di indirizzamento indiretto con autoincremento è qualcosa da tenere sempre in considerazione. Infatti l'indiretto, sia senza incremento che con incremento, impiega lo stesso numero di cicli, un ottimo caso è nell'utilizzo del Blitter, ed un esempio di questo genere lo vedremo in seguito. Il secondo metodo che abbiamo utilizzato per copiare i 1200 byte tuttavia non è proprio da buttare completamente: nel caso dovessimo fare una copia possiamo fare molto meglio, ma pensate al caso in cui dovessimo mascherare 1200 byte: siamo per forza costretti ad utilizzare un ciclo DBCC. In questi casi cercate di abusare dell'istruzione DBCC e ricordate che su un 680xx dotato di cache questi tipi di cicli sono eseguiti a velocità TURBO. Inoltre le istruzioni DBCC sono ottime anche per comparare, ecco un esempio:

```
Move.w  Len(PC),d0      ; Max lunghezza in cui cercare <> 0
Move.l  String(PC),a0
Moveq   #Char,d1        ; Carattere da cercare
FdLoop:
Cmp.b   (a0)+,d1
Dbne.s  d0,FdLoop
```

Il seguente ciclo controlla due cose contemporaneamente, infatti i cc EQ saranno settati se abbiamo esaminato tutti i Len (numero caratteri), oppure se il carattere è stato trovato, in questo caso saremmo anche in grado di dire in che posizione si trova. Vorrei a questo punto fare gli ultimi esempi sulla MOVEM ed esattamente sulla copia di zone di memoria: a differenza dell'azzeramento, qui dobbiamo prelevare dei dati e poi scaricarli, ma vediamo subito un esempio:

```
Lea     Start,a0
Lea     Dest,a1
FASTCOPY:                                   ; Impiego 13 registri
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,(a1)
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,$34(a1)      ; $34
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,$34*2(a1)    ; $34*2
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,$34*3(a1)
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,$34*4(a1)
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,$34*5(a1)
Movem.l (a0)+,d0-d7/a2-a6
Movem.l d0-d7/a2-a6,$34*6(a1)
Movem.l (a0)+,d0-d7/a2-a6
```

Innanzitutto qui abbiamo adottato la tecnica (se si può chiamarla così) dell'espansione del codice: può risultare esagerato, ma è molto efficiente. Bene, cosa abbiamo fatto ? Preleviamo 13*4 byte dalla locazione di memoria puntata ad a0, e li copiamo nella locazione di memoria puntata in a1, prestando attenzione al fatto di aumentare l'offset di a1 dopo ogni copia. Nel caso in cui vogliate espandere il codice ma, vi da fastidio vedere tutte quelle istruzioni, potete usare la direttiva REPT:

```
REPT            100
And.l           (a0)+,(a1)+
ENDR
```

Sarà poi l'assemblatore a generarle per voi. Infine vediamo un esempio legato ai registri colori:

```
Lea     $dff180,a6
Movem.l Colours(pc),d0-a5       ; carichiamo 14 longword o 28 word
Movem.l d0-a5,(a6)              ; setta 28 colori in un solo colpo!!

Colours:        dc.w    ...
```

Oppure quando all'inizio di una routine dovete caricare molti registri:

```
MOVE.L  #$4232,D0
MOVE.W  #$F20,D1
MOVE.W  #$7FFF,D2
MOVEQ   #0,D3
MOVE.L  #123456,D4
LEA     $DFF000,A0
LEA     $BFE001,A1
LEA     $BFD100,A2
LEA     Schermo,A3
LEA     BUFFER,A4
...
```

Tutto questo si può riassumere con 1 sola routine:

```
MOVEM.L VariaRoba(PC),D0-D4/A0-A4
...

VariaRoba:
dc.l    $4243           ; d0
dc.l    $f20            ; d1
dc.l    $7fff           ; d2
dc.l    0               ; d3
dc.l    $123456         ; d4
dc.l    $dff000         ; a0
dc.l    $bfe001         ; a1
dc.l    $bfd100         ; a2
dc.l    Schermo         ; a3
dc.l    Buffer          ; a4
```

Sull'istruzione MOVEM potremmo fare molti altri esempi, ma credo che abbiate capito la sua convenienza in dati casi.

Le chiamate relative al Program Counter (PC) sono più veloci di quelle normali alle label perché sono più "piccole". Infatti quelle normali devono contenere l'indirizzo lungo 32bit della label, mentre quelle (PC) contengono solo l'offset a 16 bit dal registro PC, cosa che risparmia 2 bytes e del tempo. Purtroppo, è proprio il fatto che l'offset sia a 16 bit che non permette di rendere relative al PC label più lontane di 32k in avanti o indietro. Veniamo ora ad un trucchetto per rendere relativo al (PC) tutto il programma, cosa che velocizza anche l'esecuzione. Come sapete, è possibile fare:

```
move.l  label1(PC),d0
```

Ma è impossibile rendere relativa al PC questa istruzione:

```
move.l  d0,label1
```

Come fare? Non è un problema importantissimo, ma mettiamo che abbiamo in un loop eseguito molte volte questa istruzione. Se non possiamo rendere la label relativa al PC, possiamo renderla relativa ad un comune registro indirizzi! Il metodo più evidente, è questo:

```
move.x  XXXX,label      ->      lea     label(PC),a0
move.x  XXXX,(a0)

tst.x   label           ->      lea     label(PC),a0
tst.x   label
```

Da notare, che si risparmia anche a sostituire i valori #immediati con valori caricati in registri dati, sempre che i valori siano tra -80 e +7f per permettere il MOVEQ:

```
move.l  #xx,dest        ->      moveq   #xx,d0
move.l  d0,dest


ori.l   #xx,dest        ->      moveq   #xx,d0
or.l    d0,dest


addi.l  #xx,dest        ->      moveq   #xx,d0
add.l   d0,dest
```

In particolare, se è possibile caricare tutti i registri prima di un loop, per poi risparmiare sul caricamento, si possono anche fare `MOVE.L #xx,Dx` tranquillamente, il loop senza #immediati ci ripagherà!

Esempio:

```
RoutineSchifosa:
move.w  #1024-1,d7              ; numero di loops
LoopSquallido:
add.l   #$567,label2
sub.l   #$23,label3
move.l  label2(PC),(a0)+
move.l  label3(PC),(a0)+
add.l   #30,(a0)+
sub.l   #20,(a0)+
dbra    d7,LoopSquallido
rts
```

Questa si può ottimizzare così:

```
RoutineDecente:
moveq   #30,d0          ; carichiamo i registri necessari...
moveq   #20,d1
move.l  #$567,d2
moveq   #$23,d3
lea     label2(PC),a1
lea     label3(PC),a2
move.w  #1024-1,d7              ; numero di loops
```

```
LoopNormale:
add.l   d2,(a1)
sub.l   d3,(a2)
move.l  (a1),(a0)+
move.l  (a2),(a0)+
add.l   d0,(a0)+
sub.l   d1,(a0)+
dbra    d7,LoopNormale
rts
```

Per esagerare, possiamo infine risparmiare sul numero di DBRA da eseguire:

```
RoutineOK:
moveq   #30,d0
moveq   #20,d1
move.l  #$567,d2
moveq   #$23,d3
lea     label2(PC),a1
lea     label3(PC),a2
move.w  #(1024/8)-1,d7          ; numero di loops = 128
LoopOK:

rept    8               ; replico 8 volte il pezzo...

add.l   d2,(a1)
sub.l   d3,(a2)
move.l  (a1),(a0)+
move.l  (a2),(a0)+
add.l   d0,(a0)+
sub.l   d1,(a0)+

endr

dbra    d7,LoopNormale
rts
```

Comunque, per rendere velocemente tutto PC relative, c'è un sistema. Se in un registro indirizzi stabilito, ad esempio a5, mettiamo l'indirizzo dell'inizio del programma, o comunque un indirizzo conosciuto nel nostro programma, basterà indicare la nostra label come a5+offset per trovare la label in questione. Ma dovremmo fare questo **a mano**???? Nooooo! Ecco un sistema molto veloce per fare questo:

```
S:                              ; Label di riferimento
MYPROGGY:
LEA     $dff002,A6      ; In a6 abbiamo il custom register
LEA     S(PC),A5        ; In a5 il registro per l'offset delle label

MOVE.L  #$123,LABEL2-S(A5)      ; label2-s = offset! Es: "$364(a5)"

MOVE.L  LABEL2(PC),d0           ; Qua agiamo normalmente

MOVE.L  d0,LABEL3-S(A5)         ; stesso discorso.

move.l  #$400,$96-2(a6)         ; Dmacon (in a6 c'è $dff002!!!)

...

; mettiamo di aver "sporcato" il registro A5... basterà ricaricarlo!

LEA     S(PC),A5
move.l  $64(a1),OLDINT1-S(A5)
CLR.L   LABEL1-S(A5)
```

Mi pare chiaro no? La label la potevate chiamare BAU: anzichè S:, ma credo che sia utile chiamarla S:, E:, I:, che è più corto da scrivere. L'unica limitazione è che se la label dista più di 32K dalla label di riferimento, andiamo fuori dei limiti di indirizzamento. Questo non è un problema insormontabile, infatti basta mettere una label di riferimento ogni 30K, e riferirsi alla più vicina, esempio:

```
B:
```

```
...
LEA     B(PC),A5
MOVE.L  D0,LABEL1-B(A5)
...

; passano 30K

C:

LEA     C(PC),A5
MOVE.L  (a0),LABEL40-C(A5)
...
```

Questo sistema inoltre rende difficile da disassemblare il vostro codice, nel caso che qualcuno voglia "rubarvi" le vostre routine con un disassemblatore.

Altra cosa che può esservi utile è l'uso dei bit come flag. Per esempio, se nel nostro programma abbiamo delle variabili che devono essere VERE o FALSE, ossia ACCESE o SPENTE, è inutile sprecare un byte per ognuna di esse. Basterà un bit, e risparmieremo spazio. Per esempio:

```
Opzione1        =       0
VaiDestra       =       1       ; Vai a Destra o a Sinistra?
Avvicinamento   =       2       ; Avvicinamento o Allontanamento?
Music           =       3       ; Musica Accesa o Spenta?
Candele         =       4       ; Candele accese o spente?
FirePremuto     =       5       ; qualcuno ha premuto fire?
Acqua           =       6       ; il laghetto sotto?
Cavallette      =       7       ; Ci sono le cavallette?

Controllo:
move.w  MieiFlags(PC),d0
btst.l  #Opzione1,d0
...


CambiaFlags:
lea     MieiFlags(PC),a0
bclr.b  #Opzione1,(a0)
...

MieiFlags:
dc.b    0
even
```

Comunque, se non amate i `BTST` e i `BCLR` / `BSET` / `BCHG` si può fare così:

```
bset.l  #Opzione1,d0    ->      or.b    #1<<Opzione1,d0

bclr.l  #Opzione1,d0    ->      and.b   #~(1<<Opzione1),d0

bchg.l  #Opzione1,d0    ->      eor.b   #1<<Opzione1,d0
```

Da notare l'utilità delle funzioni dell'asmone di shift "»" e "«", nonché l'eor " ".

Per terminare la sezione sulle ottimizzazioni della CPU, faccio presenti alcuni accorgimenti che velocizzano solo su 68020 e superiori, ma dato che farli non costa niente, può essere utile per veder schizzare maggiormente le nostre routines su computer più veloci. Prima di tutto, ci sono le cache, che permettono di caricare loops lunghi fino a 256 bytes, per cui dal secondo ciclo in avanti saranno letti dalla memoria interna alla cpu!!!!!!!!!!!!!! E non dalla lenta memoria (specie se chip-ram!). Di conseguenza, è bene ripetere le operazioni come abbiamo visto, nei vari loop, in modo che siano grandi sui 100-150 bytes. In questo modo, su 68020+ gireranno molto più veloci di routines in cui invece si sono messi in fila tante istruzioni quanti erano i loop da fare. Per intendersi, se abbiamo:

```
Routine1:
move.w  #2048-1,d7
loop1:
```

```
4  < blocco di istruzioni >
5  dbra    d7,loop1
6
7  Possiamo ottimizzarlo in:
8
9  Routine1:
10 rept    2048
11 < blocco di istruzioni >
12 endr
```

Che su un 68000 di base è molto più veloce, ma su un 68020 è più lento! Per fare un'ottimizzazione che sia veloce al massimo in tutti i casi:

```
1  Routine1:
2  move.w  #(2048/16)-1,d7
3  loop1:
4  rept    16
5  < blocco di istruzioni >
6  endr
7
8  dbra    d7,loop1
```

Supponiamo che il blocco di istruzioni sia lungo 12 bytes, allora 12*16 fa 192, che sta nella cache, e va velocissimo su 68020, mentre su 68000 è impercettibile la differenza con la versione con 2048 di REPT, e si risparmia anche in lunghezza dell'eseguibile. Attenzione solo a non fare i loop lunghi proprio 250 o 256 bytes, perché la cache può essere riempita solo secondo certe condizioni di "blocchi" e di "allineamento". Quindi state sempre sotto i 180-200 bytes, per sicurezza.

Altra cosa da tener presente, è che se è possibile bisogna evitare di accedere alla memoria consecutivamente. Ad esempio:

```
1  move.l  d0,(a0)
2  move.l  d1,(a1)
3  move.l  d2,(a2)
4  sub.l   d2,d0
5  eor.l   d0,d1
6  add.l   d1,d2
```

Andrebbe "riformulato" in:

```
1  move.l  d0,(a0)
2  sub.l   d2,d0
3  move.l  d1,(a1)
4  eor.l   d0,d1
5  move.l  d2,(a2)
6  add.l   d1,d2
```

Infatti, quando si accede alla memoria (specie se chip), ci sono i cosiddetti WAIT STATE, ossia tempi d'attesa prima di poterci riscrivere. Nel primo esempio, tra una scrittura e l'altra c'è un tempo morto in cui il processore attende che si possa riscrivere in ram. Nel secondo caso, invece, dopo aver scritto in ram viene eseguita una operazione tra registri, interna alla cpu, dopodiché si accede nuovamente alla chip ram, a tempo d'accesso passato. Se si accede a FAST RAM a 32bit il problema è molto meno forte, ma esiste.

Infine, i 68020+ gradiscono molto le routines e le label allineate ad indirizzi multipli di 32, ossia allineate a longword. Per allineare a 32 bit, baste un:

```
1  CNOP    0,4
```

Prima della routine o della label. Su 68000 non ci sono miglioramenti, ma ci sono su 68020+, specialmente se il codice allineato va in fast ram o in cache. Ecco un esempio:

```
1  Routine1:
2  bsr.s   rotazione
3  bsr.s   proiezione
4  bsr.s   disegno
5  rts
```

```
6
7   cnop    0,4
8   rotazione:
9   ...
10  rts
11
12  cnop    0,4
13  proiezione:
14  ...
15  rts
16
17  cnop    0,4
18  disegno:
19  ...
20  rts
```

Per le label, vedete di non accedere ad indirizzi dispari, cosa che rallenta molto, piuttosto, allineate anche queste a long:

Versione originale:

```
1   Label1:
2   dc.b    0
3   Label2:
4   dc.b    0       ; indir. dispari! il "move.b xx,label1" sarà lento!
5   Label3:
6   dc.w    0
7   Label4:
8   dc.w    0
9   Label5:
10  dc.l    0
11  Label6:
12  dc.l    0
13  Label7:
14  dc.l    0
```

Versione allineata:

```
1   cnop    0,4
2   Label1:
3   dc.b    0
4   cnop    0,4
5   Label2:
6   dc.b    0
7   cnop    0,4
8   Label3:
9   dc.w    0
10  cnop    0,4
11  Label4:
12  dc.w    0
13  cnop    0,4
14  Label5:
15  dc.l    0
16  Label6:
17  dc.l    0 ; queste 2 sono allineate sicuramente, non occorre cnop
18  Label7:
19  dc.l    0
```

Per verificare se una label è allineata a 32 bit, assemblate, poi verificate a che indirizzo si trova tale label col comando *<M>*, poi dividete l'indirizzo per 4, e moltiplicate di nuovo il risultato per 4. Se torna l'indirizzo originario, significa che è un multiplo di 4, e tutto è OK, se viene diverso significa che c'è un resto e non è un multiplo di 4. Allora mettete dei `DC.w 0` sopra l'indirizzo e provate ad allinearlo "a mano" e mandate a quel paese l'assemblatore, che è un pò suonato. Comunque, se la vostre routine funziona già al cinquantesimo, senza scatti su un a500, risparmiatevi di mettere tutti quei `CNOP 0,4` a incasinarvi il listato. Cnoppate solo i listati con routines molto pesanti che non vanno entro un frame, come routines frattali, o 3d "esagerati", eccetera.

13.4 Ottimizzazioni del Blitter

Alla fine faremo un altro esempio legato al Blitter. Questo genere di ottimizzazioni che abbiamo trattato fino ad adesso erano riferite solamente al 68000, quindi indipendenti dalla macchina a cui si riferivano, ora tratteremo delle ottimizzazioni legate all'hardware dell'Amiga, precisamente al Blitter. Come ben sapete il blitter è un potente coprocessore per lo spostamento di dati ben più veloce del 68000 di base (attenzione però che è più lento di un 68020+!). è bene trarre il massimo dal Blitter. In genere una filosofia che si adotta per il blitter è che prima inizio il trasferimento dei dati e prima finisco. Tuttavia dovete tenere sempre ben presente il bit chiamato blit nasty che è in grado di dare maggiore priorità al Blitter rispetto alla CPU, in pratica il bus per il trasferimento dei dati sarà per la maggior parte del tempo del Blitter, vediamo un esempio:

```
1  a6=$dff000
2  ; Supponiamo di aver inizializzato tutti i registri
3
4  Move.w  d0,$58(a6)       ; BLTSIZE - Il Blitter inizia
5  Wblit:
6  Move.w  #$8400,$96(a6)   ; Abilitiamo il blit nasty
7  Wblit1:
8  Btst    #6,2(a6)         ; Attendiamo che il blitter abbia finito
9  Bne.s   Wblit1
10 Move.w  #$400,$96(a6)    ; Disabilitiamo il blit nasty
11 ....
```

Questo è un caso banale, perché mentre il blitter lavora la CPU potrebbe fare qualche altra cosa, quindi quel ciclo di attesa è antiproduttivo. Infatti, in computer con sola CHIP RAM questa funzione blocca del tutto il processore, e forse non andrebbe mai usata. Ma il caso in cui possiamo e dobbiamo abilitare il blit nasty è nei casi in cui dobbiamo copiare a video un bitplane bob per bitplane, allora, poiché di solito la CPU deve attendere tra una blittata ed una altra, possiamo tranquillamente abilitare il blit nasty. Vediamo un esempio:

```
1  BLITZ:                          ; I registri sono già stati abilitati
2  Move.w  #$8400,$96(a6)  ; Abilitiamo il nasty
3  Move.l  Plane0,$50(a6)  ; Puntatore al canale A
4  Move.l  a1,$54(a6)      ; Puntatore al canale D
5  Move.w  d0,$58(a6)      ; Go Blitter!!!
6  WBL1:
7  Btst    #6,2(a6)        ; Qui la CPU deve attendere la fine...
8  Bne.s   WBL1            ; quindi il blitter deve andare al massimo!
9  Move.l  Plane1,$50(a6)  ; Puntatore al canale A
10 Move.l  a2,$54(a6)      ; Puntatore al canale D
11 Move.w  d0,$58(a6)      ; Go Blitter!!!
12 WBL2:
13 Btst    #6,2(a6)        ; Come sopra
14 Bne.s   WBL2
15 Move.l  Plane2,$50(a6)  ; Idem
16 Move.l  a3,$54(a6)
17 Move.w  d0,$58(a6)
18 WBL3:
19 Btst    #6,2(a6)
20 Bne.s   WBL3
21 Move.w  #$400,$96(a6)   ; A questo punto il nasty può anche essere
22 Rts                     ; disabilitato.
```

Questo esempio mi dà la possibilità di farvi notare una caratteristica del blitter che è quella di non modificare alcuni valori dei suoi registri, ad esempio nei registri di modulo (`BltAMod`, `BltBMod`, etc...). Troveremo al termine della blittata gli stessi valori, quindi non c'è bisogno di inizializzarli se il modulo sarà lo stesso per la prossima blittata. La stessa cosa vale per i registri tipo `BltCon0`, `BltCon1`, `BltFWM`, `BltLWM`, ma ciò non è più valido per i registri puntatori poiché questi lavorano con un indirizzamento con incremento. Ciò ci suggerisce la seguente cosa: supponiamo di avere un bob di 5 bitplane da piazzare uno per uno in un bitplane "video", quindi ogni volta carichiamo il puntatore al bitplane "video" nel registro D e il puntatore al bob in A: dopo la prima blittata il il registro D sarà caricato con lo stesso valore più una certa

quantità per puntare al bitplane successivo, ma lo stesso sarà inutile farlo con il canale A, poiché se il nostro bob è stato memorizzato in memoria come bitplane successivi, allora dopo la prima blittata il canale A punterà automaticamente al secondo bitplane del bob. Possiamo ottenere validi risultati facendo anche nel seguente modo. Riserviamo una zona di memoria con tutti i valori da passare ai registri del blitter (nel nostro caso la zona parte da DataBlit). Quindi in alcuni registri indirizzi carichiamo gli indirizzi dei registri del blitter in modo tale da poterci accedere più velocemente, e copiamo i dati preconfezionati per far partire il blitter, accedendo direttamente ai registri della CPU. Vediamo un esempio:

```
Lea     $dff002,a6      ; a6 = DMAConR
Move.l  DataBlit(pc),a5 ; quindi a5 punta a una tabella di valori
; precalcolati

; Carichiamo ora i registri indirizzi

Lea     $40-2(a6),a0    ; a0 = BltCon0
Lea     $62-2(a6),a1    ; a1 = BltBMod
Lea     $50-2(a6),a2    ; a2 = BltApt
Lea     $54-2(a6),a3    ; a3 = BltDpt
Lea     $58-2(a6),a4    ; a4 = BltSize
Moveq   #6,D0           ; d0 costante per il controllo dello stato
; del blitter.
Move.w  (a5)+,D7        ; Numero di blittate
Move.w  #$8400,$96-2(a6) ; Abilitiamo il nasty
BLITLOOP:
Btst    d0,(a6)         ; Attendiamo come sempre la fine di qualche
Bne.s   BLITLOOP        ; operazione.
; Prima di guardare qui sotto facciamo un
; osservazione, se in a0 ho il valore $40000
; ed eseguo le istruzione in tre casi distinti
; a)Move.b #"1",(a0)
; b)Move.w #"12",(a0)
; c)Move.l #"1234",(a0)
; otterrò la seguente cosa:
;         (a)   (b)     (c)
; $40000        "1"     "1"     "1"
; $40001        "0"     "2"     "2"
; $40002        "0"     "0"     "3"
; $40003        "0"     "0"     "4"
; Ora faremo una cosa del genere...
Move.l  (a5)+,(a0)      ; $dff040-42 cioè Bltcon0-Bltcon1
Move.l  (a5)+,(a1)      ; $dff062-64 cioè BltBMod-BltAMod
Move.l  (a5)+,(a2)      ; $dff050 - Canale A
Move.l  (a5)+,(a3)      ; $dff054 - Canale D
Move.l  (a5)+,(a4)      ; $dff058 - BLTSIZE... START!!
Dbra    d7,BLITLOOP     ; Questo per d7 volte.
```

In questo esempio abbiamo adoperato varie tecniche di ottimizzazione, di cui abbiamo già parlato, in ogni caso vediamone qualcuna. Innanzitutto quando dobbiamo eseguire un gran numero di volte un ciclo ed all'interno c'è una operazione che coinvolge una costante (cioè un dato immediato), conviene mettere questo valore in un registro che non si userà nel ciclo, quindi effettuare l'operazione che coinvolge questo valore direttamente col registro che lo contiene, evitando un accesso alla memoria. Nel nostro caso abbiamo adoperato questa strategia caricando il valore del bit da testare, per controllare se il blitter aveva terminato il suo compito, nel registro d0. In pratica abbiamo adottato una delle prime regole di cui vi ho parlato all'inizio cioè cercare sempre di tenere i valori nei registri. Inoltre, abbiamo caricato $dff002 come base e non $dff000. Questo viene fatto spesso, per eliminare il tempo usato nel waitblit a calcolare l'offset:

```
Btst    #6,2(a6)        ; a6 = $dff000
```

è più lento di:

```
btst    d0,(a6)         ; a6 = $dff002, d0 = 6
```

Basta ricordarsi di mettere un -2 prima di (a6) per ottenere l'offset giusto:

```
$54-2(a6)       ; BltDpt
$58-2(a6)       ; BltSize
$96-2(a6)       ; DmaCon
...
```

è importante che il waitblit sia veloce, in quanto prima si "accorge" che la blittata è finita, prima si comincia quella dopo! Per questo, evitate di chiamare il waitblit con un BSR, bensì mettetelo sempre sul posto, anche ripetendolo ogni volta che serve.

Lo stesso discorso che abbiamo fatto ora lo abbiamo applicato anche per i registri del blitter caricandoli nei registri della CPU, evitando accessi alla memoria (in pratica alla memoria ci accediamo comunque per inizializzare il blitter, ma evitiamo ogni volta di prelevare l'indirizzo dalla memoria). Inoltre abbiamo usato un trucco che chiunque programmi giochi o demo adopera, cioè invece di tenere in memoria le dimensioni del bob per poi calcolarsi il valore del bltsize, teniamo direttamente il valore del bltsize, questo lo abbiamo fatto tramite la tabella DataBlit. Tuttavia, come vi ho accennato sopra, mentre il blitter lavora il 68000 può fare qualche altra cosa, ad esempio se il blitter sta cancellando una zona di memoria, il 68000 da buon cristiano può dargli una mano, ad esempio:

```
btst    #6,2(a6)
WaitBlit:
btst    #6,2(a6)
bne.s   WaitBlit
Moveq   #-1,d0
Move.l  d0,$44(a6)              ; -1 = $ffffffff
Move.l  #$9f00000,$40(a6)
Moveq   #0,d1
Move.l  d1,$64(a6)
Move.l  a0,$50(a6)
Move.l  a1,$54(a6)
Move.w  #$4414,$58(a6)          ; Il blitter inizia a pulire...
Move.l  a7,OldSp
Movem.l CLREG(pc),d0-d7/a0-a6   ; Puliamo i registri
Move.l  Screen(pc),a7           ; Indirizzo della zona da cancellare
Add.w   #$a8c0,a7               ; andiamo alla sua fine (+$a8c0)

Rept            1024            ; Il 68000 inizia a pulire
Movem.l d0-d7/a0-a6,-(a7)       ; Pulisci 60 bytes per 1024 volte
EndR

Lea     $dff000,a6
Movea.l OLDSP(pc),a7
Rts

CLREG:
ds.l    15
```

Come vedete, qua il blitter e la cpu puliscono "in contemporanea" mezzo schermo per uno. Naturalmente in questo caso il nasty bit non deve essere settato, o la cpu non può pulire in pace.

Il miglior metodo per aumentare le prestazioni del proprio programma rimane però quello di migliorare i propri algoritmi, molto spesso. Non pensiate ad esempio che implementare in assembly un pessimo algoritmo di ordinamento, come il Bubble Sort, sia più veloce del miglior algoritmo di ordinamento, come il Quick Sort, implementato in C. Se il vostro algoritmo non vuole proprio girare più velocemente anche dopo aver adoperato le migliori tecniche di ottimizzazione, bè, allora cancellatelo e riscrivetelo completamente con un algoritmo migliore **in partenza**. Ed anche se siete in possesso del miglior algoritmo, cercate sempre di ottimizzarlo in modo da farlo girare su delle macchine anche non veloci, non come nel mondo dei PC dove un programmatore di un 486 si sente soddisfatto solo se il suo codice gira velocemente sulla propria configurazione. Che ci vuole a fare routines veloci, se poi sulla confezione del gioco o del programma si legge: CONFIGURAZIONE MINIMA: PENTIUM 60Mhz con 8MB di RAM.

CAPITOLO 14

LEZIONE 14 - FONDAMENTI DI ACUSTICA E AUDIO DIGITALE

Autore: Alvise Spanò

Come tutti senz'altro saprete, il suono altro non è che *un'onda*, ovvero, secondo la definizione fisica, la *"propagazione di una perturbazione in un mezzo"*; nel caso dei suoni che siamo abituati ad ascoltare, la perturbazione (onda) viene inizialmente emessa dalla vibrazione della materia (molecole e/o atomi), ed il mezzo è l'aria (un'onda è una oscillazione, un continuo scambio di energia cinetica e potenziale, quindi meccanica, tra molecole e/o atomi (fisicamente è la variazione di pressione tra particelle di materia), e pertanto *necessita* di materia per esistere e diffondersi: nel vuoto, ad esempio, non si propaga alcuna onda sonora, né alcun altro tipo di vibrazione tra due corpi distinti e separati (le radiazione elettromagnetiche soltanto - almeno sinora, le uniche di cui se ne è alla conoscenza - riescono a muoversi anche nel vuoto grazie alla loro duplice natura fisica sia di corpuscolo dotato di massa - seppur piccolissima - (fotone) sia di onda)). Il modello fisico adottato per descrivere questo continuo cedimento di energia da un punto all'altro del mezzo presenta il grafico della funzione goniometrica SENO (= sen = sin), ovvero di una SINUSOIDE.

Funzione d'onda $y = f(x \pm v * t)$ Ad esempio l'equazione d'onda armonica:

$$y = a\sin(k(x \pm vt)) = a * \sin(k * (x \pm v * t))$$

- $y =$ *variabile dipendente* in un grafico cartesiano bidimensionale (x,y), l'ordinata y rappresenta le "quote" di ogni punto x dell'oscillazione.
- $a =$ *coefficiente di amplificazione dell'onda* come ben saprete, $-1 \leq \sin(x) \leq 1$ (seno di x (x = qualsiasi numero reale) compreso tra -1 e 1, estremi inclusi), quindi per ottenere un'oscillazione che vari da $-a$ ad a ($-a <= \sin(x) <= a$) è necessario moltiplicare $\sin(x)$ per un numero reale a.
- $k =$ *frequenza dell'onda* variando questo parametro si varia la frequenza, e, in modo inversamente proporzionale, il *periodo* dell'onda, ovvero l'intervallo minimo, lungo l'asse della

variabile indipendente (in questo caso x) per cui la sinusoide è ciclabile, ovvero il minimo intervallo dopo il quale l'onda assume le medesime caratteristiche. Introduciamo a questo punto il concetto di *lunghezza d'onda* (= spazio percorso nel periodo = distanza tra due creste di cicli adiacenti), ed approfondiamo quello di frequenza: numero di volte in cui vengono assunte le stesse caratteristiche dall'onda in una data unità di tempo, ovvero quante volte al secondo viene letto il periodo (= cicli al secondo); di solito si considera il minuto secondo [s] come unità di tempo e l'Hertz [Hz = $[s]^{-1} = \frac{1}{[s]}$ come unità di misura della frequenza.

- x = *variabile indipendente* in un grafico cartesiano bidimensionale (x, y), l'ascissa x rappresenta un punto lungo una dimensione spaziale rettilinea prestabilita in un dato istante; x appartiene ai numeri reali e, teoricamente, non ha limitazioni, ovvero abbraccia tutta la retta provocando nel piano considerato (x, y) una sinusoide che prosegue all'infinito sia a sinistra che a destra dell'origine degli assi $O(0, 0)$: ciò fa facilmente intuire come la funzione seno goda di una periodicità. Per esempio, se $k = 1$, il periodo dell'onda è di 360 gradi (= $2 * \pi$ radianti) e la frequenza è 1 Hz; se k = 2, il periodo è 180° (= π rad) e la frequenza è pari a 2 Hz; e così via.

- v = *velocità di propagazione* indica la velocità in senso cinematico con cui si sposta nello spazio un punto dell'onda. Da notare che v = LUNG_ONDA $*$ FREQ.

- t = *istante di tempo* tenete presente che $v * t = s == spazio$, ed s è la distanza tra due punti corrispondenti della stessa perturbazione presa in t diversi, pertanto, aggiungendo/-sottraendo s a/da x si ottiene nel tempo una propagazione, un movimento nello spazio dell'onda. è da notare che con $(x + s)$ l'onda si muoverà a sinistra, e con $(x - s)$ a destra lungo l'asse delle ascisse.

P.S.: mi scuso per la frettolosità delle spiegazioni e l'assenza di dimostrazioni, ma non mi sembra questa l'occasione adatta per dilungarsi eccessivamente su argomenti che non riguardano direttamente l'assembler ed il coding in generale; per cui vi prego di prendere così come sono le suddette delucidazioni e non preoccupatevi troppo se non avete capito nel profondo la fisica delle onde: non vi servirà alla fin fine per inserire una musica in un gioco o demo.

N.B.: Tenete presente che x e y rappresentano due qualsiasi caratteristiche del fenomeno di propagazione. Nel caso di onde sonore, noi considereremo x e y come due dimensioni spaziali che descrivono su di un piano l'onda esaminandone una sezione.

Ora siamo in grado di trasportare quanto appreso dagli accenni di fisica generale dei fenomeni di propagazione in acustica vera e propria, definendo il concetto di ARMONICA: un suono - tra parentesi, inesistente in natura e riproducibile solo con strumenti elettronici, quali il computer appunto - che presenta la forma d'onda (= grafico (x,y) della perturbazione lungo tutta la sua durata) di una sinusoide. Possiamo dire che l'armonica è il modello del *suono base*, che unendosi a molti altri crea tutti i suoni *non puri*. Sempre la fisica distingue in un suono 3 qualità affinché possa essere descritto:

1. *altezza* che si distingue di suoni *puri* (costituiti da una sola armonica – inesistenti in natura) e *naturali* (costituiti da più armoniche sovrapposte in uno stesso intervallo di tempo – alcuni di quelli che esistono in natura presentano migliaia di armoniche di periodo diverso), e ne qualifica la frequenza, alterando la quale vengono prodotte le varie note.

2. *intensità* che può essere vista come una sorta di "volume" del suono, e, nel caso di armoniche, è direttamente proporzionale all'amplificazione a (valore assoluto delle quote Y delle creste) della sinusoide.

3. *timbro* che qualifica la forma d'onda del suono a prescindere dai precedenti due parametri, quindi descrive sostanzialmente lo strumento musicale o, più generalmente, ciò per cui noi distinguiamo un suono da un altro indipendentemente dalla sua altezza e intensità.

Lasciamo perdere le varie formule per calcolare l'intensità sonora ed i livelli di pressione e di intensità [deciBel = dB], che comunque non toccano direttamente la musica elettronica ma solo l'acustica come ramo della fisica che si occupa della diffusione dei suoni nell'ambiente e della progettazione di apparecchi in grado di riprodurre (altoparlanti, ecc.) o captare (microfoni, ecc.) suoni, e soffermiamoci sull'altezza ed il timbro.

> Intanto per cominciare, il timbro, di per se, è un parametro inesistente: il computer non distingue i suoni, e converte i dati digitali che ha in memoria in segnali analogici (che trasferiscono intensità elettriche e non bit) secondo una frequenza di lettura ed un dato valore di sottoamplificazione, non curandosi della differenza tra suono e rumore – due concetti che soltanto noi uomini abbiamo introdotto con significati diversi e riconducibili al senso estetico.

Del timbro come parametro, dunque, non si può propriamente parlare – perlomeno a livello elettronico –, anche se esistono sofisticati algoritmi di identificazione e confronto del timbro di forme d'onda differenti che potrebbero tornare utili a chi è intenzionato a programmare routine di riconoscimento vocale, ad esempio; ad ogni modo, non è certo questa la sede adatta a discutere di tanto complesse applicazioni del mondo della sintesi e della elaborazione sonora.

Passiamo quindi al parametro più importante - per quanto ci riguarda - : l'altezza dei suoni è direttamente legata, nel caso di suoni NATURALI, alla frequenza dell'armonica fondamentale (*frequenza fondamentale*) che li distingue, e, nel caso di suoni *puri*, alla frequenza dell'*unica* armonica che li costituisce. Facciamo un esempio: abbiamo un suono puro (armonica) di periodo (=durata) X; esso può essere emesso a qualsiasi frequenza, dipende solo dalla "*velocità di lettira* del suono da parte dell'emittente: per esempio, se questo "legge" l'armonica per tutto il suo periodo 2 volte al secondo la frequenza sarà pari a 2 Hz; in un secondo momento, si presenta pure il problema della diffusione acustica del dato suono nell'aria, che, però, è presto risolto: la velocità di propagazione del suono nell'aria è di 380 m/s, e quindi, sapendo che VELOCITà = LUNGH.ONDA * FREQ., è estremamente semplice ottenerne la lunghezza d'onda, che coincide, in termini di dimensioni spaziali, al periodo (che è espresso in secondi). A questo punto entra in gioco un nuovo parametro, per trattare gli strumenti di generazione del suono: la velocità di lettura, o *velocità / frequenza di campionamento*. Ora però, per proseguire, è necessario spiegare come gli strumenti elettronici trattano il suono e come lo processano prima di passarlo all'amplificatore.

Tramite l'uso di un campionatore (digitalizzatore audio) ed un software adeguato, è possibile convertire i suoni provenienti da una sorgente sonora (microfono, CD, ecc.) in dati numerici (campioni digitali) ognuno dei quali descrive la "quota" di un intervallo (di una "minuscola fettina" – per parlare in termini scientifici –) lungo le ascisse del grafico (x,y) della forma d'onda. Più sono i campioni che "catturiamo", più definita e vicina alla realtà fisica sarà il suono digitale. Ad esempio, se abbiamo un suono naturale (quindi, non armoniche) della durata di 2 secondi (che assumiamo come periodo, visto che non è possibile individuare un intervallo minimo inferiore per cui l'onda si cicli), e siamo in grado di ricavarne la frequenza fondamentale, è sufficiente campionare in memoria con una DOPPIA frequenza di campionamento (teorema di Nyquist, spiegato più avanti), per ottenere una riproduzione fedele e definita acusticamente del suono; se, ad esempio, la frequenza fondamentale è di 2 kHz (= 2000 Hz), dovremmo registrare 2000 campioni al secondo, per un totale di 4000 campioni (2000 camp/s * 2 s = 4000 camp).

Ogni campione (= sample), in memoria occupa 8 bit (= 1 byte), nell'Amiga, che adotta una definizione sonora ad 8 bit, appunto. Col numero espresso ad 8 bit è possibile descrivere quote Y comprese tra -128 (= -$(2^8)/2$ = -$2^{(8-1)} = -2^7) e 127 (= (2^8)/2 - 1 = 2^{(8-1)} - 1 = 2^7 - 1$ (lo zero ha segno positivo, in binario: in effetti i numeri positivi (da 0 a 127) sono 128 come quelli negativi), estremi inclusi, di ogni singolo campione, per un totale di 256 (= 2^8) valori esprimibili.

N.B.: è importante notare che la forma d'onda oscilla tra il I (+) ed il II (-) quadrante, divisi dall'ascissa di quota 0; NON considerate -128 come 0 e +127 come 255: NON è possibile traslare l'onda sul I quadrante e rendere tutto positivo, *i conti non tornerebbero nè al chip sonoro, nè ad un'eventuale elaborazione del suono per effetti speciali via software*.

Considerate ogni sample come un byte ad 8 bit con segno (= MSB = bit 7)

È importante sottolineare come sintesi digitale ad un numero di bit maggiore di 8 offrirebbero una qualità sonora superiore, a parità di frequenza di campionamento. Per esempio, i lettori CD leggono sample a 16 bit (= 2 byte = 1 word), il che significa che il range delle quote varia da -32768 (= $-(2^16)/2 = -2^(16-1) == -2^15) e 32767 (= (2^16)/2 - 1 = 2^(16-1) - 1 = 2^15 - 1$), per un totale di 65536 (= 2^16) valori esprimibili.

Ciò non vuol dire, però, che il valore 32767 (picco positivo) del CD corrisponde ad una quota maggiore di quel sample rispetto al valore 127 dello stesso campione reso ad 8 bit: l'uscita sonora sarà uguale, solo che a parità di range fisico la sintesi a 16 bit offre molta più definizione (in sostanza, ad 8 bit ho 256 numeri per esprimere un suono, compreso tra due picchi positivo e negativo fisicamente costanti, che però a 16 bit viene sintetizzato con una gamma di valori molto superiore (65536) e quindi *con più precisione, con meno approssimazione delle quote*). In un certo senso possiamo affermare che gli 8 bit del primo esempio corrispondono agli 8 bit alti (15:8) dei 16 bit del secondo, e gli 8 bit bassi corrispondono ad una sorta di approssimazione dopo una virgola fittizia posta tra il byte alto e quello basso della word del sample.

Torniamo ora a parlare della frequenza di campionamento, citando un celebre ed importante – tanto quanto complicato per quanto riguarda la dimostrazione – che enuncia che "*la risposta di frequenza è pari alla metà della frequenza di campionamento* (teorema di Nyquist): sostanzialmente, significa che se noi campioniamo a 10 kHz, verranno riprodotti fedelmente solo i suoni con frequenza minore od uguale a 10/2 = 5 kHz (ecco spiegato il misterioso *doppio* scritto poco sopra circa la frequenza di campionamento necessaria per campionare il dato suono, conoscendone la frequenza fondamentale). È fondamentale campionare i suoni ad una frequenza adeguata per non sentire l'*aliasing*, che taglia le frequenze al di sopra della metà della frequenza di campionamento rendendole con uno sgradevole effetto di "disturbato".

Anche se il Paula (per la cronaca, nome proprio del chip sonoro dell'Amiga) adotta una precisione digitale a soli 8 bit, è possibile riprodurre suoni ugualmente di buona qualità campionando alle frequenze giuste ed evitando l'aliasing, anche se, comunque, non si può raggiungere la qualità di un CD, che campiona 16 bit a 44.1 kHz (44100 Hz) per ottenere una risposta di frequenza che spazia da 20 Hz a 22 kHz circa, che corrisponde circa al range delle frequenze udibili dall'orecchio umano (soggettivo: qualcuno potrebbe arrivare fino a 20 kHz ca.)

Colgo l'occasione per dire che i suoni di frequenza (fondamentale, sott'inteso) minore di 20 Hz vengono chiamati *infrasuoni*, e quelli di frequenza maggiore di 20-22 kHz *ultrasuoni*: entrambi **non** sono udibili dall'uomo.

Comunque, la maggior parte dei suoni naturali non ha una frequenza fondamentale superiore ai 15-16 kHz; è dunque sufficiente campionare a 32 kHz al massimo per riprodurre fedelmente quasi tutti i suoni esistenti? Ebbene, no ! Per un motivo molto semplice: poco sopra è stato spiegato che i suoni naturali sono costituiti da molte armoniche tra le quali ne è individuabile una fondamentale: potrebbe anche accadere che la frequenza fondamentale (che noi utilizziamo di solito per calcolare quella di campionamento) sia effettivamente la frequenza dell'armonica di periodo minore (quindi, di frequenza maggiore); in tal caso tutte le armoniche di frequenza maggiore di quella che noi assumiamo come fondamentale verrebbero tagliate e riprodotte con aliasing, abbassando sensibilmente la qualità sonora globale.

Sarebbe opportuno quindi, campionare a frequenza doppia rispetto alla frequenza dell'armonica più alta che compone il dato suono naturale

Dunque, l'intensità contraddistingue il "volume" del suono, ed essa **non** è costante rispetto alla frequenza: a frequenze molto alte o molto basse è necessario amplificarlo per essere percepito con la stessa intensità rispetto a suoni di frequenza media dall'orecchio, che l'evoluzione biologica (conseguenza dell'abitudine) lo ha evidentemente portato a sentire meglio quelli più diffusi in natura; cosa contraddistingue invece l'altezza di un suono? In termini puramente musicali, è presto detto: le *note*. Come senz'altro saprete le note musicali formano scale di 7 note per *ottava*, ognuna delle quali comincia con la nota DO (C, per la notazione anglosassone) e finisce con il SI (B, per la notazione anglosassone - il LA è la A); ad ogni ottava si raddoppia la frequenza, quindi ogni DO è di frequenza doppia rispetto a DO precedente (notate dunque che l'incremento di frequenze non è rettilineo, ma *esponenziale* in base 2). All'interno dell'ottava i rapporti tra le note della scala sono i seguenti:

DO	RE	MI	FA	SOL	LA	SI	(DO)
1	9/8	5/4	4/3	3/2	5/3	15/8	2

Nel caso in cui bisogni campionare suoni che poi verranno usati come strumenti in programmi di editing musicale (tipo SoundTracker, NoiseTracker o ProTracker), sarebbe sufficiente campionare ad una frequenza doppia rispetto alla frequenza fondamentale del suono, che, se si campiona da uno strumento musicale, corrisponde alla frequenza della nota suonata, almeno per praticità: se, ad esempio, necessitiamo di un pianoforte per la composizione di un modulo, possiamo campionare il LA3 (LA della terza ottava) a 880 Hz (LA3 = 440 Hz) e comunicare al tracker che la frequenza di campionamento corrisponde al LA3, penserà lui, dopo, a calcolare le frequenze giuste relative a quella di campionamento in base alle note che poniamo sullo spartito con quello strumento. Ora, sicuramente, vi porrete una domanda: ma basta campionare a 880 Hz, per non ottenere un aliasing? La risposta è no. Come abbiamo detto prima, bisognerebbe campionare al doppio della frequenza dell'armonica più acuta per riprodurre fedelmente la timbrica del pianoforte, ma è assai complicato ricavare tale frequenza (per non dire impossibile). Cosa fare, allora? Beh, provare e riprovare a campionare a varie frequenze (onestamente, ben più alte di 880 Hz) finché si ottiene una riproduzione ottimale dello strumento a quella nota e comunicare al tracker la frequenza di lettura del sample per tale nota. Come vedete, dunque, la faccenda è più complicata nella pratica che nella teoria!

Dopo questi inevitabili (e – spero – interessanti) cenni di audio digitale, passo alla spiegazione più specifica dell'hardware sonoro dei chip Original ed AA dell'Amiga (il Paula è l'unico chip custom che non ha mai subito miglioramenti dall'uscita del primo Amiga (1985, per la cronaca)). L'hardware presenta 4 canali DMA dedicati alle 4 voci del chip sonoro; queste 4 voci sono totalmente indipendenti e sono raggruppate a 2 a 2 per cassa ottenendo le voci 1+4 per

la via sinistra e 2+3 per la destra in stereo. Tutte e 4 le voci, inoltre, possiedono propri registri hardware:

```
AUDxLCH $dff0y0 =       Locazione dei dati da leggere (word alta)
AUDxLCL $dff0y2 =       Locazione dei dati da leggere (word bassa)
AUDxLEN $dff0y4 =       Lunghezza del DMA (in word)
AUDxPER $dff0y6 =       Periodo di campionamento in lettura
AUDxVOL $dff0y8 =       Volume
AUDxDAT $dff0ya =       Dato del canale (2 byte = 2 sample alla volta)
```

N.B.: Per ogni 'x' sostituite un numero da 0 a 3, corrispondente alla voce desiderata; per ogni 'y' sostituite un numero esadecimale da $a a $d relativo alle voci da 0 a 3.

`AUDxLCH-AUDxLCL` Costituiscono il valore di latch, non il puntatore del DMA ai dati, pertanto, una volta impostato, non incrementa come accade per i plane, per gli sprite o per i canali del blitter, ma è simile ai registri di locazione del copper, il valore dei quali viene automaticamente riinserito nei registri interni di puntatore quando ce n'è bisogno. * Visto che questi due registri a 16 bit sono adiacenti, è comodo impostarli con un singolo `MOVE.L` del 68000 del tipo:

```
1 MOVE.L #miosample,AUDxLCH *.
```

N.B.: d'ora in poi, con `AUDxLC` mi riferirò alla coppia dei due registri, ad una sorta di registro di locazione unico a 32 bit.

`AUDxLEN` Esprime la lunghezza in word del sample da suonare. Se, per esempio, abbiamo in memoria un sample di 500 byte, bisogna impostare questo registro (di uno dei 4 canali desiderato) con un valore di 250. N.B.: Come per il blitter, scrivendo 0 questo registro vengono letti 128 kB di sample.

`AUDxPER` Questo registro serve a specificare la frequenza di lettura del DMA in modo un pò bizzarro – apparentemente –, ma che torna comodo e veloce all'hardware: bisogna impostarlo con il PERIODO DI CAMPIONAMENTO di ogni singolo campione del suono, un valore che esprime il tempo (in cicli di CLOCK del DMA di sistema = 3546895 Hz (PAL), 3579545 Hz (NTSC)) che il DMA deve attendere (funziona da decrementatore: -1 per ciclo di clock) prima di trasferire un'altro campione. Ecco una formula per calcolare il valore da inserire in questo registro, data la frequenza di campionamento (che è molto più pratica da gestire): PER = CLOCK / freq. [Hz] Per esempio, se dobbiamo campionare un LA3 armonica – ammesso che ne troviamo un sorgente naturale...– di frequenza 440 Hz, di periodo 1 secondo, dobbiamo adottare una frequenza di campionamento di 880 Hz, per cui ecco il periodo di campionamento da inserire nel registro `AUDxPER` per leggere alla giusta frequenza in 1 secondo il sample in memoria: PER = 3546895 / 880 = 4030 (PAL) N.B.: I dati audio vengono fetchati dal DMA in 4 slot del color clock (16bit=2 sample per canale) per linea di scansione orizzontale. Le linee di scansione PAL sono 312.5 (312=SHF, 313=LOF) per raster, e ci sono 50 raster al secondo; per cui, la frequenza massima di lettura (leggendo a tutti i cicli assegnati disponibili) sarebbe = 2 BytePerLinea * 312.5 * 50 = 31300 Hz circa: **ma questa velocità è solo teorica**, in quanto è impossibile impostare un giusto periodo di campionamento che coincida perfettamente con i cicli assegnati al DMA audio ad ogni linea di raster: potrebbe verificarsi la fine di un conteggio del periodo dal DMA nel bel mezzo di una linea di scansione (o, sfiga totale, il ciclo dopo lo slot assegnato alla data voce), e l'hardware è costretto ad attendere la linea dopo per avere i dati da suonare, mentre, se il periodo di campionamento è breve, la successiva fine di conteggio potrebbe

avvenire nella linea stessa, in assenza di nuovi dati. In sostanza, il periodo minimo deve permettere all'hardware di percorrere **almeno** un'intera linea di raster: il suono non esce quando viene letto dal DMA, ma quando termina il conteggio interno dal valore del periodo in AUDxPER: il DMA audio legge i dati solo durante i cicli assegnati (peraltro ad altissima priorità – come quelli del DMA dei disk drive –, per evitare distorsioni e rallentmenti dovuti a "sovraffollamento" di canali – vedi: "bit plane che rompono sempre") e li conserva in AUDxDAT fino alla fine del del periodo di campionamento, *che può avvenire in qualsiasi momento*, e viene generato il suono; *per questo motivo non si può abbassare il periodo minimo di campionamento oltre 123 (=28836 Hz): per permettere al DMA di leggere comunque almeno un'altro dato prima di suonare, alla linea dopo*. A questo punto la domanda è d'obbligo: "Come mai si pone 123 come periodo minimo, quando i cicli di clock per linea (ovvero il numero di decrementi) sono 226.5 (226=LOF, 227=SHF)?". Ecco la risposta: prima è stato accennato che il DMA trasferisce 16 bit (=2 byte) "al colpo" per voce, per cui possiede 2 sample suonabili ad ogni linea, e, suonato il primo, può suonare anche il successivo durante la stessa linea di raster, per il fatto che non lo ha già letto; con 123, infatti, possono accadere, al massimo, 2 fine-conteggi durante una stessa linea, ed il problema non si pone. Il periodo minimo teorico, dunque, dovrebbe essere di 227 (per tenerci larghi) / 2 = 114 (sempre approssimando per eccesso), che coincide più o meno (la corrispondenza tra periodo e frequenza di campionamento NON è biunivoca: c'è sempre una certa approssimazione) con la frequenza massima teorica di 31300 Hz circa. Ma, come si è detto sopra, non è raggiungibile con precisione dall'hardware. *123 è il periodo minimo impostabile = 28836 Hz*

AUDxVOL In questo registro va specificato il volume dell'emissione del suono nel relativo canale con valori compresi tra 0 e 64 (inserendo '64' non vi è alcune diminuzione di dB = volume massimo).

CONSIGLIO!!! Quando campionate cercate di sfruttare tutta la fascia di valori da -128 a 127 anche per suoni a bassa intensità al fine di avere sempre la massima precisione, e, tutt'al più, abbassate il volume in questo registro.

AUDxDAT è il buffer momentaneo che il DMA utilizza prima che i dati vengano spediti ai convertitori D/A (Digitale/Analogico) e che vengano emessi i segnali all'esterno dell'Amiga. Esso contiene 2 byte di dati audio (il DMA trasferisce 16 bit alla volta dalla RAM – ecco spiegato il motivo per cui l'AUDxLEN deve essere espresso in word !) che vengono inviati 1 ad 1 al DAC (Digital-Analogical Converter).

SCONSIGLIO!!! è anche possibile impostare questi registri con la CPU quando il DMA è spento e far suonare ugualmente il computer :(.

Un esempio di impostazione dei registri per suonare un sample di 23 kB ad una frequenza di 21056 Hz posto in memoria alla locazione $60000 (chip RAM) a volume massimo in stereo, con la voce 2 e la 3 (terzo e quarto canale):

```
1  PlaySample:
2          lea     $dff000,a0      ; base dei chip custom in a0
3          move.l  #$60000,$c0(a0) ; punta AUD2LC a $60000
4          move.l  #$60000,$d0(a0) ; punta anche AUD3LC a $60000
5          move.w  #11776,$c4(a0)  ; AUD2LEN = 23 kB = 23*1024 = 23552 B [...]
6          move.w  #11776,$d4(a0)  ; anche AUD3LEN = [...] = 23552/2 = 11776 word
7          move.w  #168,$c6(a0)    ; AUD2PER = 3546895/21056 = 168
8          move.w  #168,$d6(a0)    ; imposta AUD3PER come AUD2PER
9          move.w  #64,$c8(a0)     ; volume massimo per AUD2VOL
```

```
10          move.w  #64,$d8(a0)     ; volume massimo anche per AUD3VOL
11          move.w  #$800c,$96(a0)  ; accende il DMA dei canali 2 e 3 in DMACON
```

Passiamo ora a spiegare cosa accade quando si accendono i DMA dei canali (oltre al fatto che – chiaramente – si sente il sample...):

1. Il valore contenuto in `AUDLC` viene inserito nei registri interni di puntamento ed il DMA comincia a trasferire nei registri dati 2 byte alla volta. Da adesso il registro `AUDLC` **può** anche venire cambiato: l'hardware, appena finito di trasferire tutto il sample, ricomincierà daccapo (LOOPANDO ALL'INFINITO).

2. Appena inserito il valore di `AUDLC` nei registri interni viene sparato un interrupt di LIVELLO 4, che nei registri `INTENA` e `INTREQ` si suddivide in 4 sottointerrupt, uno assegnato ad ognuno dei 4 canali audio:

LIVELLO IRQ	BIT INTENA/INTREQ	CANALE
4	10	3
4	9	2
4	8	1
4	7	0

   Grazie a questi interrupt è possibile, ad esempio, impostare un nuovo sample da suonare appena finito di suonare quello corrente semplicemente puntando i registri locazione ad un altro sample, ed ottenendo un perfetto collegamento fra i due (a patto che le due forme d'onda rispettivamente terminino e comincino similmente).

3. alla fine del trasferimento ricomincia tutto dal punto (1). *I registri non vengono mai alterati*.

Certo, tutto questo è interessante – voi direte –, ma come fare per far suonare al computer una canzone di 10 minuti senza campionarsi decine di megabyte di dati?

Proprio per questo scopo sono stati inventati i Tracker: programmi atti a scrivere musica che richiedono che soltanto gli strumenti di base siano campionati ricavando le varie note variando la frequenza di lettura degli stessi. Sono inoltre dotati di un editor dove comporre lo spartito suddiviso in 4 tracce (una per voce), ognuna delle quali può suonare qualsiasi strumento in qualsiasi momento, ma pur sempre uno alla volta (in totale si può ottenere un massimo di 4 strumenti effettivamente in contemporanea).

Sarebbe assai complicato spiegare qui tutte le varie possibilità di un tracker, pertanto vi invito a procurarvene uno (tipo il ProTracker: attualmente il miglior tracker a 4 tracce – ebbene si, ne esistono anche a più tracce, che mixano in tempo reale le note di più tracce in un'unica voce; purtroppo, però, tale procedimento è estremamente lento e non potrebbe essere adottato per suonare la musica di un demo o di un videogioco, visto che la macchina, in simili casi, ha ben altro da fare che perdere tutto il tempo a suonare...). La "filosofia" dei tracker, comunque, non cambia: tutti forniscono delle – cosiddette – music routine, che sono dei sorgenti asm che sfruttando spesso interrupt di livello 6 (collegati al CIAB) o loop di attesa sincronizzati col pennello elettronico, e suonano in tempo reale i moduli (song + samples = spartito + strumenti) creati con il relativo tracker (o compatibili, ovvero, che salvano la struttura del modulo allo stesso modo). Adattare tali routine ai propri sorgenti è semplicissimo: in linea di massima – ognuna ha le sue convenzioni: leggetevi i .doc del vostro tracker - è sufficiente lanciare una subroutine di inizializzazione che imposta gli interrupt ed i canali DMA – alcune anche i timer del CIAB; il CIAA resta intatto visto che viene usato dall'Exec per temporizzare i processi/task – e lanciare la

subroutine di play ad ogni raster (vi conviene farlo all'inizio del codice di interrupt del vertical blank - se siete sotto sistema operativo aggiungete un server di interrupt 5 (`VBLANK`, livello 3) a priorità alta, per fare tutto durante l'intervallo di vertical blank, prima che i plane comincino a fetchare ed a rallentare tutto); prima di quittare il vostro demo/gioco - o programma che sia – **ricordate** di lanciare la subroutine di restore degli interrupt, dei DMA e dei timer all'OS.

Un altro importante paragrafo sull'hardware audio dell'Amiga riguarda la modulazione del suono proveniente dal DMA delle 4 voci. Cos'è la *modulazione*? Avrete senz'altro udito in moltissime canzoni l'effetto di*dissolvenza audio* (il progressivo e lento abbassarsi del volume): ebbene, questo semplice effetto è un particolare tipo di modulazione.

> *La MODULAZIONE consiste nell'alterare uno o più parametri di un suono durante ed oltre il suo periodo; i parametri in questione sono, ovviamente,* intensità *(ampiezza) ed* altezza *(frequenza)*.

A quali effetti corrispondono – a livello di percezione sonora – modulazione in ampiezza e modulazione in frequenza ? La prima, come abbiamo già accennato, trova una comune applicazione nelle dissolvenze solitamente al principio ed al termine di un brano musicale; un familiare esempio di modulazione in frequenza è lo slide sulle corde di una chitarra (od uno strumento a corda): in sostanza, una sfumata fusione di note adiacenti partendo da una data frequenza ed arrivando ad un'altra passando gradualmente per tutte quelle intermedie con una certa velocità (od addirittura una certa accelerazione). È anche possibile modulare sia in ampiezza che in frequenza contemporaneamente, ottenendo uno strano effetto riconducibile ad un fenomeno di esperienza quotidiana: *l'effetto Doppler*.

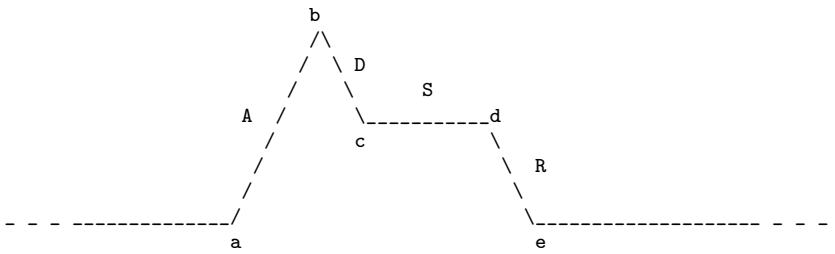
Brevemente, consiste nel cambiamento (appunto, modulazione) di intensità ed altezza dei suoni provenienti da una fonte in moto relativo rispetto all'ascoltatore: quando state camminando per strada, notate che il rumore delle macchine che vi si avvicinano e poi vi sorpassano non mantiene i medesimi parametri nelle diverse posizioni dell'auto (fonte) rispetto a voi (ascoltatore) ma, innanzitutto, si fà più alto di volume in modo inversamente proporzionale alla distanza tra voi e la fonte, e, se ci badate bene, l'intensità non è l'unica a mutare nel tempo: anche la frequenza del rumore emesso dal motore è minore quando la macchina è distante.

Non penso sia il caso di riportare l'equazione che descrive il fenomeno in funzione della velocità dei due corpi e della distanza, in quanto il problema non tocca da vicino l'argomento "modulazione su Amiga"; equazione che, comunque, potrete facilmente trovare in qualunque libro di fisica o di acustica generale anche delle scuole medie superiori.

Passando, appunto, alla modulazione sull'Amiga, sono costretto a deludervi subito: sebbene il Paula possegga dei particolari modi operativi per modulare i suoni provenienti da un canale sia in ampiezza che in frequenza, non si usa mai utilizzare questa soluzione hardware perché presenta una terribile restrizione: per modulare intensità ed altezza il DMA deve leggere dalla RAM dei valori da inserire rispettivamente nei registri `AUDxVOL` e/o `AUDxPER` mentre un altro DMA legge i valori veri e propri del sample da suonare e poi distorcere; tale processo ha la limitazione che il DMA che legge dalla tabella dei valori di modulazione deve essere uno dei canali audio, per cui per modulare, ad esempio, il suono letto dal canale 0 sia in frequenza che in ampiezza siamo costretti ad usare i canali 1 e 2 per leggere le rispettive tabelle, col risultato di sprecare 3 canali per generare un solo suono modulato. Tutti gli effetti di modulazione utilizzati dai tracker sono gestiti dalla CPU, che imposta "di cattiveria" i registri di volume e periodo della voce desiderata mentre il DMA legge ignaro il suo sample. Così facendo, non viene sprecato alcun canale, anche se la CPU resta per un pò impegnata a calcolare in tempo reale gli effetti sonori.

L'Amiga NON possiede nemmeno un sintetizzatore FM (Frequency Modulation) in grado di creare timbri diversi partendo da una stessa forma d'onda. Questi applicano modulazioni sia in

ampiezza che in frequenza secondo 4 parametri chiamati ADSR, dalle iniziali delle 4 fasi principali di un suono sintetizzato: Attack, Decay, Sustain, Release. Il grafico di questa modulazione è il seguente:



La prima fase è l'Attack, che consiste nel portare il volume e/o la frequenza dell'onda da 'à a 'b' (ovvero da 0 ad un valore di picco massimo); dopodiché, il grafico scende per il tratto del Decay fino ad una quota 'c', alla quale si stabilizza per la durata del Sustain; infine ritorna a 0 da 'd' ad 'è con il Release. "Giocando" acutamente con la posizione dei punti 'à,'b','c','d' ed 'è e con le durate delle varie fasi è possiblile generare un'infinità di suoni anche partendo dal sample di una banale armonica. Purtroppo, queste nozioni non vi saranno utili per la programmazione del chip sonoro dell'Amiga, perciò passiamo alla descrizione di quei bit che mai vennero impostati nella storia di Amiga e del suo hardware... :) Il registro in questione è il famigerato `ADKCON` (*$dff09e*), che possiede pure una copia di lettura (`ADKCONR`) all'indirizzo *$dff010*:

```
bit  -  7:      USE3PN  Usa il canale 3 per non modulare nulla
        6:      USE2P3  Usa il canale 2 per modulare il PERIODO del 3
        5:      USE1P2  Usa il canale 1 per modulare il PERIODO del 2
        4:      USE0P1  Usa il canale 0 per modulare il PERIODO dell' 1
        3:      USE3VN  Usa il canale 3 per non modulare nulla
        2:      USE2V3  Usa il canale 2 per modulare il VOLUME del 3
        1:      USE1V2  Usa il canale 1 per modulare il VOLUME del 2
        0:      USE0V1  Usa il canale 0 per modulare il VOLUME dell' 1
```

Avrete senz'altro capito come funziona la faccenda: se, ad esempio, dovete modulare in ampiezza il canale 2, potete farlo solamente utilizzando l'1 come lettore dei valori da inserire nel registro `AUD2VOL`, per cui, dovrete puntare i canali ai relativi dati e dare loro una frequenza di lettura. La modulazione, comunque, è un effetto semplice ma importante, che deve essere simulato via CPU - come già detto - per non occupare qualcuno dei già pochi canali audio di Amiga...

Concluderei permettendomi di affermare che la conoscenza di acustica digitale e del funzionamento dei chip sonori in generale, non è fondamentale come quella dell'hardware grafico o dell'asm di una CPU, ed è certo che la padronanza di questi è necessaria a chiunque, audiofili compresi; è altrettanto vero, però, che la vera cultura in materia prevede anche la conoscenza approfondita della teoria del suono digitale, che tanto viene decantata in questi ultimi tempi, quanto è oggetto di ignoranza dalla maggior parte della gente, che ragiona proprio come quei "coder" che snobbano – per non dire "saltano" – completamente le fonti di informazione su tale argomento perché "tanto, le music routine basta chiamarle dall'interrupt...".

14.1 Le replay routines sofisticate *(Autore: Fabio Ciucci)*

A proposito di tali music rutines, per ora abbiamo visto solo quella standard fornita con il protracker, ma ce ne sono anche altre più sofisticate. Vedremo ora una delle migliori, il `player6.1a`,

che richiede anche un programma di conversione (il `p61con`, in questo disco), con cui dobbiamo trasformare un modulo normale in uno ottimizzato per la replay routine.

NOTA: Questo player è copyright dell'autore:

```
Jarno Paananen / Guru of Sahara Surfers.
```

Quindi, se userete la sua replay routine in un prodotto commerciale, ad esempio in un gioco, dovete ricevere la sua autorizzazione scritta e dargli qualcosa (in marchi finlandesi!) come percentuale. Già si arrabbierà con me che non ho incluso tutto l'archivio!!!

Questo player ha moltissime opzioni, vediamo intanto di farci la cosa più semplice: suonare un modulo come abbiamo fatto con la routine standard fornita assieme al programma protracker.

Le cose da fare sono queste:

1. Convertire il modulo nel formato P61, usando l'utility "P61CON". Questa utility richiede la `reqtools.library` e la `powerpacker.library` nella directory `libs:` per funzionare. Nelle preferences del programma, non toccate niente, lasciando settata solo l'opzione "tempo". Annotatevi al salvataggio l'USECODE, che andrà specificato nel listato all'equate `USE = .....` Questo serve per risparmiare codice.

2. In questo modo, abbiamo ottenuto il modulo convertito **non compresso**. Nonostante ciò, spesso il modulo si accorcia per l'ottimizzazione che viene fatta automaticamente.

3. Ora basta fare come con le routines precedenti: chiamare `P61_Init` prima di suonare, poi `P61_Music` ad ogni fotogramma, e `P61_End` alla fine. L'unica richiesta in più è l'abilitazione dell'interrupt di livello 6.

Vediamo un esempio pratico in `Lezione14-10a.s`.

Noterete come sia più veloce di quella standard, ma anche che usa il timer A del CIAB e l'interrupt di livello 6 ($78). Ci sono poi degli equates, di cui occorre conoscere il significato:

```
1 fade  = 0        ;0 = Normal, NO master volume control possible
2 ;1 = Use master volume (P61_Master)
```

Questo va messo ad 1 se si vuol controllare il volume per fare un fade, agendo sulla label `P61_Master`. Vedremo dopo un esempio. Se non ci serve tale opzione, mettiamo 0, in modo da risparmiare codice, infatti questi equate non sono altro che degli assemblaggi condizionati che usano le direttive dell'assemblatore `ifeq`, `ifne`, `endc...`

```
1 jump = 0         ;0 = do NOT include position jump code (P61_SetPosition)
2 ;1 = Include
```

Anche questa opzione va lasciata a zero, se non si usa la routine di salto ad una specifica posizione del modulo. Vedremo dopo un esempio.

```
1 system = 0       ;0 = killer
2 ;1 = friendly
```

Questa opzione è bene lasciarla a 0, se si fa codice "malvagio" e si usa la `startup2.s`. State solo attenti quando caricate da dos!!! (dovete anche lasciare l'interrupt $78 (level6) al suo posto, senza ripristinare quello di sistema, nel caso carichiate con questa replay routine attiva!).

```
1 CIA = 0          ;0 = CIA disabled
2 ;1 = CIA enabled
```

Questa opzione va tenuta a 0 per usare la routine in modo "standard". Se si mette ad 1, non occorrerà più chiamare `P61_Music` ad ogni fotogramma, perché la temporizzazione avverrà totalmente col CIAB. Vedremo un esempio.

```
exec = 1        ;0 = ExecBase destroyed
;1 = ExecBase valid
```

Qua va lasciato 1, dato che in $4.w lasciamo l'execbase valido... non siamo mica dei maniaci!!! E la startup che la facciamo a fare?

```
opt020 = 0      ;0 = MC680x0 code
;1 = MC68020+ or better
```

Questo è chiaro: se il vostro gioco/demo è AGA only, potete mettere questo ad 1, altrimenti lasciatelo a zero. Attenti a non metterlo ad 1 quando non serve!!!!!! **Solo se il gioco/demo va solo su AGA (quindi 68020+)**.

```
use = $2009559  ; Usecode (mettete il valore dato dal p61con al salvataggio
; diverso per ogni modulo!)
```

Qua il commento spiega tutto... annotatevi sempre l'usecode su un foglietto (senza perdere il foglietto, naturalmente), e mettetelo qua. Serve per far assemblare solo le routines degli effetti usati nel modulo, risparmiando spazio. Mettere -1 significa assemblare tutto (puah!).

Il programma di conversione permette anche di comprimere il modulo, ma questo fa perdere un pò la qualità. Quindi vi sconsiglio di compattarli... A meno che non dobbiate fare una 40k intro e abbiate le spalle al muro in termini di spazio, è sempre bene usare il modulo "normalmente" convertito. Comunque, il programma permette di scegliere quali sample compattare e quali no... e di ascoltare con le nostre orecchie se si perde troppo di qualità!!!

Ecco cosa si deve fare per comprimere e risuonare un modulo compresso:

1. Convertire il modulo nel formato P61 compresso, con "P61CON"- Nelle preferences del programma, occorre attivare l'opzione "pack samples". Da notare che potete scegliere quali sample comprimere e quali no. Ecco cosa vedrete per ogni sample:

   Original Suona il sample originale (Stop con tasto destro mouse)

   Packed Suona il sample come verrebbe compresso. Se notate che si perde troppo in qualità, ripensateci...!

   Pack Segna questo sample come "da comprimere"

   Pack rest Compatta tutti gli altri sample da qua in avanti

   Don't pack Non compattare questo sample

   Don't pack rest Non conpattare tutti gli altri sample (da qua in avanti)

   Annotatevi al salvataggio l'USECODE, come sempre. In più, annotatevi anche il "sample buffer length"!!!!!!

2. In questo modo, abbiamo ottenuto il modulo convertito e **compresso**.

3. Ora ci sono 2 cose in più da fare: innanzitutto, il modulo ha i sample compressi, che vanno scompattati in un buffer. Per fare ciò, occorre fare 2 cose: il buffer, lungo quanto indicato dal programma come "sample buffer length", e mettere il suo indirizzo in `a2` prima di chiamare `P61_Init`, che provvederà alla decompressione. Per il resto (play ed end) è uguale.

Vediamo in pratica il tutto:

```
        movem.l d0-d7/a0-a6,-(SP)
        lea     P61_data,a0     ; Indirizzo del modulo in a0
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
******************
>>>>>   lea     samples,a2      ; modulo compattato! Buffer destinazione per
******************              ; i samples (in chip ram) da indicare!
        bsr.w   P61_Init        ; Nota: impiega alcuni secondi per decompress!
        movem.l (SP)+,d0-d7/a0-a6
```

Per il modulo e il buffer, ecco le modifiche:

1. Il modulo non occorre più che sia caricato in chip ram:

```
        Section modulozzo,data  ; Non occorre sia in chip ram, perchè è
                                ; compresso e sarà scompattato altrove!
P61_data:
        incbin  "P61.stardust"  ; Compresso, (opzione PACK SAMPLES)
```

2. Il buffer deve essere caricato in CHIP RAM, e lungo quanto specificato:

```
        section smp,bss_c

samples:
        ds.b    132112  ; lunghezza riportata dal p61con
```

Come noterete, oltre a perdere qualità nei samples, si usa anche più memoria, in quanto abbiamo in più il buffer, anche se il modulo è più corto. Vediamo un esempio in `Lezione14-10b.s`.

Ora che abbiamo visto le 2 implementazioni principali, possiamo vedere tutte le varianti. Innanzitutto l'opzione CIA, che si abilita con l'equate. Potete vederene 2 esempi in `Lezione14-10c.s` e `Lezione14-10d.s`. Per finire, vediamo l'uso di 2 optional:

Il fade audio: basta attivare l'equate "FADE", e agire sulla apposita label `P61_Master`, che va da 0 a 64. Esempio in `Lezione14-10e.s`.

La possibilità di saltare a posizioni arbitrarie del modulo: basta attivare l'equate `JUMP`, e chiamare la routine `P61_SetPosition`, con la posizione nel registro `D0`. Esempio in `Lezione14-10f.s`.

Ci sarebbero anche altri optional, che possiamo riassumere nelle preferences:

```
Two files:              Questa opzione salva separatamente i sample e il
                        song in 2 file. Può servire se usiamo più moduli
                        con gli stessi sample...

P61A sign:              mette il segno P61A all'inizio del modulo... può
                        servire solo a rendere più facile ai malintenzionati
                        ripparlo!!! Non settatelo mai!

No samples:             Serve quando si salvano molti moduli che hanno gli
                        stessi sample: la prima volta si mette "two files",
                        e si salvano i moduli e il primo song. Poi si mette
                        questa opzione e si salvano tutti gli altri songs.

Tempo:                  Per far usare l'opzione "Tempo" al player.

Icon:                   Se si vuol salvare un icona assieme al modulo

Delta:                  Compressione a 8-bit anzichè a 4-bit (ho notato
                        che non cambia quasi niente... bah!)

Sample packing:         Da settare per la compressione dei samples con
                        l'algoritmo delta a 4-bit (PERDENDO DI QUALITà!!!).
```

Buon ascolto a tutti!

CAPITOLO 15

LEZIONE 15 - IL CHIPSET AGA

Questa è l'attesissima lezione sul nuovo chipset AGA, presente nell'A1200 e nell'A4000. Quando uscì l'Amiga4000, alla fine del 1992, un mio amico lo comprò subito, e in pratica andai a stare a casa sua, tanto che lo ho usato di più io che lui. In quei primi mesi disassemblai le copperlist del sistema operativo e pezzi interi di KickStart, perché la defunta Commodore non dava a nessuno documentazione sull'AGA. Strano, ma vero. Comunque a forza di prove cominciai a capire qualcosa, ma mancava anche l'iffconverter AGA, e dovetti "convertire" a mano le figure da IFF a RAW. L'unico programma che era in grado di visualizzare schermate AGA in quel tempo era il nuovo DeLuxePaint, per cui caricavo una figura a 256 colori, poi in multitasking mi caricavo l'asmone e cercavo in memoria la figura .raw e la copperlist per salvarle. In un secondo tempo ricaricavo il raw, lo puntavo nella copperlist e incrociavo le dita. Comunque non riuscii ad essere il primo a fare una demo AGA, la prima la fecero gli *ABYSS*, una piccola demo che però visualizzava i fatidici 256 colori. Niente di eccezionale, ma erano stati i primi. Più o meno però ero allo stesso punto degli Abyss nella scoperta dell'AGA, e non mi scoraggiai. Era ormai Febbraio 1993, ero quasi pronto per una intro con un logo in 640*256 a 256 colori che ondeggiava con la fluidità di 1/4 di pixel (usando il nuovo `BPLCON1`), quando uscì la prima **vera** demo AGA, ossia *PLANET GROOVE* dei *TEAM HOI*.

Chiamai subito la loro BBS in Olanda, lasciando un messaggio al coder, *Rhino*. Da quel giorno cominciò un rapporto di (costosi) messaggi tra di noi, dove ci scambiavamo le ultime scoperte e la funzione degli ultimi bit sconosciuti. Poco prima era uscito *ZOOL AGA*, che in realtà non aveva proprio nulla di AGA, per cui l'unico codice decentemente AGHIZZATO era la demo di Rhino, il quale si programmò anche un Iffconverter AGA (il primo uscito), che usai con molto piacere.

Dato che non esisteva alcuna documentazione sull'hardware del 1200, e che di conseguenza non si vedevano demo né giochi AGA, misi insieme le informazioni che avevo scoperto assieme a Rhino in un `AGADOC.TXT`, ma quando ero quasi pronto per distrinuirlo nelle BBS uscì un piccolo testo, `hard1200.txt`, opera di *Yragael*, un coder francese. In questo testo c'erano alcune cose che non sapevo, ma mancavano molte cose che sapevo io. Chiamai un poco di BBS in Francia e riuscii a trovarlo, e seppi che stava programmando anche lui un iffconverter per l'AGA, che salvava anche gli sprites larghi 64 pixel. Quell'IffConverter storico è presente nel disco di utility del corso. Misi tutte le informazioni insieme, e feci un gran bel testo, farcito anche di informazioni

sul 68020. Questo testo circolò per le bbs, e anche per i party. In teoria ero pronto per fare una demo aga, e infatti ne feci una per lo SMAU dell'ottobre 1993 a Milano, ma in realtà si tratta di uno slideshow "molto tecnico" più che di una demo (lo programmai in 2 settimane, faccio sempre le cose all'ultimo minuto!). Comunque c'erano figure a 256 colori in hires interlacciato, fade AGA a 24 bit (come al cinema!), nonché una figura in HAM8 (credo sia la prima figura in HAM8 visualizzata in una demo!!!), e un effetto di fade "incrociato" a 24 bit che ha avuto molto successo. Oggigiorno escono moltissime demo AGA, e giochi come *SUPER STARDUST* o *BRIAN THE LION* sfruttano finalmente le nuove possibilità. Nonostante abbia programmato la prima demo AGA italiana, poi mi sono "fermato" e non ho più fatto niente, tanto che l'ultima demo che ho fatto era per A500.

Perché? Non lo so. Comunque col mio `AGADOC.TXT` e qualche consiglio ho contribuito alla programmazione della seconda demo AGA italiana, ossia *IT CAN'T BE DONE*, programmata da *EXECUTOR/RAM JAM*, che ha texture mapping vario. Mentre Executor ha messo nella sua demo un dovuto ringraziamento per l'aiuto che gli ho dato, ben poche tra le prime demo AGA straniere contengono saluti per me, ma credo che molti abbiano usato il mio prezioso (almeno in quei tempi) agadoc. Qualche tempo dopo la Commodore cominciò a mandare il manuale delle specifiche AGA alle software house, per cui qualcuno lo "rubò" e lo trascrisse (furono i *COMBAT 18*), di conseguenza il mio agadoc divenne meno "esclusivo".

Questa era la storia della scoperta dell'aga, dove posso considerarmi tra i primi 10 pionieri, anche se mi chiedo tuttora se ne sia valsa la pena, dato che poi mi sono passati tutti avanti leggendo qualche mese dopo la documentazione bella e pronta. Vi propongo una traduzione in italiano del mio primo AGADOC, dato che lo scrissi in inglese.

Innanzitutto occorre precisare che per visualizzare immagini AGA non occorre usare istruzioni 68020, si potrebbe fare una demo aga con istruzioni tutte del 68000 base, dato che le differenze stanno nel COPPER. Questo significa che potete programmare anche col *TRASH'M'ONE*, che non supporta istruzioni 68020, ma ovviamente se le userete conviene passare al *TFA ASMONE 1.25* presente nel disco di Utility: tra l'altro ha l'help in linea dei registri aga, come nel TRASH'M'ONE, solo che anzichè usare *<=C>* occorre usare *<=R>*, per esempio per vedere il registro *$dff106* (`BPLCON3`) basta digitare *<=R 106>*. Abbiamo già visto come "disabilitare" l'aga:

```
1       move.w  #0,$1fc(a5)             ; FMODE - disabilita fetch 64/32 bit.
2       move.w  #$c00,$106(a5)          ; BPLCON3 - disattiva palette 24 bit
3       move.w  #$11,$10c(a5)           ; BPLCON4 - palette normale.
```

Ebbene, ora dovremo vedere come abilitare tutto! Cominciamo con un sommario delle nuove possibilità, giusto per invogliarvi a sapere come usarle: la palette ora anziché essere a 12 bit, ossia 4096 colori, è stata portata a 24 bit, ossia 16 milioni. Mentre prima per ogni componente RGB si poteva scegliere un numero da 0 a 15, ora si può scegliere un numero tra 0 e 255. Dunque: 16*16*16=4096 colori possibili nel vecchio modo OCS e ECS, mentre 256*256*256=16777216 colori tra cui scegliere nell'AGA.

Per esempio, prima si potevano fare 16 toni di grigio al massimo, ossia si immetteva nei registri colore *$0000*, *$0111*, *$0222*, *$0333* ecc, mentre ora si possono fare 256 toni di grigio. Anche i bitplanes disponibili sono aumentati, infatti ora possono essere anche 8, ossia 256 colori. (8 bit=256 possibilità). Esiste anche un modo HAM8 speciale, con 262144 colori "teorici" sullo schermo, ma alcune limitazioni (lievi "sbavature"), simili all'HAM6 normale. HAM8 sta per HAM con 8 bitplanes, mentre HAM6 è l'HAM normale a 6 bitplanes. Il nuovo Dual Playfield può avere fino a 4 bitplanes per playfield (16 colori un playfield e 16 l'altro), e il banco dei 16 colori nella palette di 256 è selezionabile indipendentemente per ogni playfield.

Come se non bastasse, anche gli sprites si sono "evoluti". Ricordate il limite di larghezza di 16 pixel? Ebbene gli 8 sprites ora possono essere larghi anche 32 o 64 pixel ciascuno, e si può scegliere se devono essere in lowres o in HIRES, indipendentemente dalla risoluzione dello

schermo. Per esempio, si possono visualizzare 8 sprites in hires, larghi 64 pixel, su uno schermo in lowres a 256 colori. Sprite Attached sono disponibili sempre. Gli sprites pari e dispari possono usare il loro banco indipendente di 16 colori dalla palette dei 256 totali. Comunque uno sprite non attached ha sempre un massimo di 3 colori + sfondo e uno attached 15 colori + sfondo. Una novità è pure quella che gli sprites possono apparire anche nei bordi, ossia fuori dalla finestra DIWSTART-DIWSTOP, mentre normalmente non potevano. Per attivare questa possibilità basta settare il bit 1 del *$dff106* (BPLCON3) Come se non bastasse, il posizionamento orizzontale è stato portato a 32ns, ossia anziché fare 320 "scatti" per percorrere lo schermo orizzontalmente, ora possono fare passi più piccoli, anche di un quarto di pixel, come se lo schermo fosse in 1280*256, e si facesse 1 pixel alla volta. Questo permette di far ondeggiare gli sprite come nessuna scheda SUPER VGA del pc msdos può fare.

La possibilità di uno scrolling fluidissimo a passi di 1/4 di pixel è stata implementata anche per i bitplanes, si tratta di bit "extra" nel *$dff102*, nel buon vecchio BPLCON1. è possibile fare decine di livelli in parallasse con lo scrolling più incredibile della storia dei computer. Il nuovo *$dff102*, oltre a permettere scroll a "scattini" di 1/4 di pixel alla volta, ora può scorrere fino ad un massimo di 64 pixel, anziché 16. Anche se ci interessa in modo minore, è possibile già dall'ECS gestire schermi a 31khz, ossia per monitor multisync. Col chipset AGA è possibile "deinterlacciare" gli schermi a 15Khz, bitplanes e sprite compresi, per i monitor SUPERVGA. Le demo e i giochi comunque di solito sono in PAL! Tutte queste novità comunque se non "azionate" non interferiscono sulla compatibilità con il vecchio chipset, come avrete verificato eseguendo i sorgenti OCS/ECS delle precedenti lezioni.

In particolare occorre azzerare il *$dff1fc* (FMODE) e il bit 0 del BPLCON0. Questo bit lo abbiamo sempre tenuto azzerato nelle lezioni precedenti. Settandolo diventano operativi altri bit nel BPLCON3 (*$dff106*), tra cui BRDRSPRT, quello degli sprite fuori dai "bordi". Per rilevare le collisioni con i bitplanes 7 ed 8, che non sono supportati dal CLXCON, esiste il CLXCON2 (*$dff10e*), che si resetta scrivendo nel vecchio CLXCON, permettendo una corretta segnalazione delle collisioni nei giochi OCS. Non si sa al momento attuale se eventuali Amiga che usciranno nel futuro supporteranno l'AGA o soltanto l'ECS, si è detto che forse supporteranno solo l'OCS/ECS in emulazione e avranno un sistema grafico diverso. Comunque per i probemi che ci sono stati per vendere la Commodore ecc. il ritardo ha portato ad allontanare l'uscita di questi nuovi modelli, per cui l'AGA durerà per molti anni, e questo probabilmente porterà a supportarlo nelle eventuali nuove macchine Amiga.

Comunque c'è anche il CD32 che supporta l'AGA. Se volete programmare giochi per CD32 considerate che ha 2 porte joystick che supportano 11 "bottoni", per cui dovete adattare il codice a questo joy. Altre differenze del CD32 sono 1Kb di flash RAM, dove si può salvare l'HIGH SCORE o le password dei giochi, nonché il chip AKIKO, che dovrebbe essere in grado di convertire grafica da Chunky a Planar, ma pare non sia velocissimo. Convertire da Chunky (modo video come la VGA) a Bitplanes amiga serve per la grafica in texture mapping, vedi DOOM sul PC MSDOS. Più avanti forse ci faremo un nostro DOOM.

Come prima cosa occorre vedere se il computer ha l'AGA, abbiamo già visto la routine per il detect:

```
1           LEA     $DFF000,A5
2           MOVE.W  $7C(A5),D0      ; DeniseID (o LisaID AGA)
3           MOVEQ   #100,D7         ; Controlla 100 volte (per sicurezza, dato
4                                   ; che il vecchio denise da valori casuali)
5   DENLOOP:
6           MOVE.W  $7C(A5),D1      ; Denise ID (o LisaID AGA)
7           CMP.B   d0,d1           ; Lo stesso valore?
8           BNE.S   NOTAGA          ; Non è lo stesso valore: Denise OCS!
9           DBRA    D7,DENLOOP
10          BTST.L  #2,d0           ; BIT 2 azzerato=AGA. è presente l'aga??
11          BNE.S   NOTAGA          ; no...
12          ST.B    AGA             ; si... settiamo il flag "AGA" allora.
```

```
13 NOTAGA:                          ; non AGA... o OCS/ECS o il futuro AAA...
14         ...
```

15.1 La nuova palette a 24 bit

Ok, ora vediamo in pratica come visualizzare 128 o 256 colori, come fare delle sfumature col copper a "24 bit", eccetera. Per prima cosa è importante capire come funziona la nuova palette, perché poi per il resto si tratta soltanto di settare qualche bit qua e là per aggiungere bitplanes o allargare gli sprites. Abbiamo detto che per ognuna delle 3 componenti ROSSO, VERDE e BLU si può dare un valore da 0 a 255 anziché da 0 a 15. Se prima per settare il giallo occorreva mettere *$F* di rosso, *$F* di verde e 0 di blu, ora occorre *$FF* di rosso, *$ff* di verde e *$00* di blu. Fin qua tutto chiaro. Se prima dovevamo mettere in *$dff180* il valore *$0ff0* per il giallo (*$0RGB*), ora dove mettiamo *$00FFFF00*? Nel registro, che è una word, non c'entra *$00ffff00*, ossia *$00RRGGBB*. I progettisti hanno trovato un modo per mantenere la compatibilità con l'OCS e per far entrare 256 colori a 24 bit nei vecchi 32 registri a 12 bit!! Vediamo intanto come hanno risolto il primo problema, ossia quello di far entrare 1 colore *$RRGGBB*, ad esempio, nel COLOR0 (*$dff180*). Facciamo questa considerazione: se avessimo il colore a 12 bit *$f32*, come sarebbe l'equivalente a 24 bit? Naturalmente *$f03020*. Ora, si può notare che i colori a 4 bit normalmente usati nell'OCS/ECS sono i 4 bit alti, o in altri termini il nibble alto, dei colori ad 8 bit dell'AGA. Ed è proprio così! Se azzeriamo i registri dell'AGA e mettiamo nel COLOR0 o in un'altro registro colore un valore, cambiamo i 4 bit alti delle 3 componenti RGB, lasciando azzerati i 4 bit bassi, per cui il colore risultante è lo stesso di quello di un OCS. Avrete intuito che per settare un colore a 24 bit occorre metere separatamente i bit alti (*$RxGxBx*) nel *$dff180*, poi "scambiare" qualcosa e mettere i bit bassi (*$xRxGxB*) sempre nel *$dff180*. Facciamo un esempio: abbiamo il colore a 24 bit *$437efa*, ossia RED = *$43*, GREEN = *$7e*, BLU = *$fa*. Ecco come facciamo in copperlist:

```
1         dc.w    $180,$47f       ; metto i nibble alti
2         "scambio"
3         dc.w    $180,$3ea       ; metto i nibble bassi
```

Per ora abbiamo messo "scambio", vediamo in pratica cosa si fa per scambiare la funzione del *$dff180* da "ricettore di nibble bassi" a "ricettore di nibble alti" del colore a 24 bit. Per selezionare i bit alti, abbiamo messo il valore *$c00* nel BPLCON3 (*$dff106*) infatti nell'emulazione ECS i registri colore valgono sempre come ricettori di bit alti del colore. In teoria si potrebbe mettere anche *$000* nel *$dff106*, perché settare i bit 10 e 11 serve solo nel modo DUAL PLAYFIELD a resettare delle cose che vedremo dopo. Si intuisce dunque che quando "un certo bit" del *$dff106* è azzerato, i registri colore ricevono i bit alti, quando è settato invece ricevono i bit alti. Può sembrarvi contorto spezzare i valori RGB in questo modo, ma dato che la palette per le figure la salva l'iffconverter già pronta, non c'è di che lamentarsi.

Inoltre si possono fare delle routines che fanno copperlist o che "spezzano" i colori in questo modo. Il bit del *$dff106* che si occupa di "scambiare" la funzione dei registri colore è il nono, detto LOCT. Siccome quando scriviamo nei nibble alti (MSB) quelli bassi (LSB) sono azzerati per la compatibilità con OCS/ECS, quando si vuol settare un colore a 24 bit occorre caricare prima i bit alti, poi quelli bassi.

Ecco uno schemino del colore *%RRRRrrrrGGGGggggBBBBbbbb* (binario), dove le lettere maiuscole sono i bit alti della tonalità, quelle minuscole i bassi.

```
BIT#    11,10, 9, 8     7, 6, 5, 4     3, 2, 1, 0
----    -----------    -----------    -----------
LOCT=0  R7 R6 R5 R4    G7 G6 G5 G4    B7 B6 B5 B4
LOCT=1  r3 r2 r1 r0    g3 g2 g1 g0    b3 b2 b1 b0

R = RED    G = GREEN    B = BLUE
```

Si può dire che i registri colore AGA hanno due facce, e si gira la faccia settando o azzerando il bit 9 di *$dff106*. Il bit 9 settato produce il valore *$200* (*%0000001000000000*). Per cui si può sostituire "scambio" con *$106*, *$200*:

```
        dc.w    $106,$000       ; Seleziono i nibble alti
        dc.w    $180,$47f       ; metto i nibble alti
        dc.w    $106,$200       ; Seleziono i nibble bassi
        dc.w    $180,$3ea       ; metto i nibble bassi
```

Molti settano anche i bit 10 ed 11, che come abbiamo detto servono solo per il dual playfield, comunque non fanno male:

```
        dc.w    $106,$c00       ; Seleziono i nibble alti
        dc.w    $180,$47f       ; metto i nibble alti
        dc.w    $106,$e00       ; Seleziono i nibble bassi
        dc.w    $180,$3ea       ; metto i nibble bassi
```

Dunque *$c00* per selezionare i bit alti, poi *$e00* per selezionare i bit bassi. Naturalmente se si debbono settare 10 colori non mettiamo ogni volta il `BPLCON3` tra un colore e l'altro, ma semplicemente:

```
        dc.w    $106,$c00       ; Seleziono i nibble alti

        dc.w    $180,$47f       ; metto i nibble alti di tutti i colori
        dc.w    $182,$123
        dc.w    $184,$456
        dc.w    $186,$789
        dc.w    $188,$abc
        dc.w    $18a,$def

        dc.w    $106,$e00       ; Seleziono i nibble bassi

        dc.w    $180,$3ea       ; metto i nibble bassi di tutti i colori
        dc.w    $182,$111
        dc.w    $184,$444
        dc.w    $186,$888
        dc.w    $188,$434
        dc.w    $18a,$abc
```

Ora è il momento di verificare in pratica se tutto questo funziona, proviamo a fare delle "barrette" tipo la lezione3, ma AGA: vedetevi `Lezione15a.s`

Noterete che è moooolto lungo scriversi la copperlist in AGA. Quindi per certe sfumature o cose ripetitive si fa prima a farsi una routine. In particolare vedetevi `Lezione15b.s` per fare delle sfumature.

15.2 I nuovi modi a 128 e 256 colori

Vediamo ora, invece, come è possibile "caricare" 256 colori se i registri colore sono soltanto 32. Infatti sappiamo che ogni registro colore ha 2 facce, che vedono una i nibble bassi e l'altra i nibble alti, ma sappiamo fare al massimo una figura a 32 colori, anche se tali colori sono scelti da una palette di 24 bit. Ebbene, c'è un altro trucco, anche questo nel *$dff106*.

I registri colore dovrebbero essere 256, e ce ne sono 32, ossia un ottavo di quelli che ci servono. Se azzerando il *$dff106* si accede ai primi 32 colori, si intuisce che ci deve essere un bit che, se settato, fa accedere ai registri dei colori dal 33 al 64, scrivendo sempre nel *$dff180-$dff1be*. Infatti ci sono 8 banchi con 32 registri colore ciascuno, e si deve decidere (con i bit 13,14 e 15 del *$dff106*) a quale degli 8 banchi accedere scrivendo nei registri colore:

```
------- bit --- $dff106 (BPLCON3) -----------------------------------

        15      BANK2   | Con questi 3 bit si seleziona uno degli 8 banchi
        14      BANK1   | di registri per accedere ai 256 colori AGA
```

```
        13      BANK0   |
------------------------------------------------------------------------
```

La selezione del "banco" funziona in modo analogo alla selezione del bitplane nel BPLCON0 (*$dff100*), infatti questi 3 bit sono letti "assieme" e, a seconda del numero che contengono, selezionano il banco corrispondente:

```
valore dei 3 bit  - banco di colori corrispondente - valore del \$dff106

        000             COLORE  00 - COLORE  31         $c00  / $e00
        001             COLORE  32 - COLORE  63         $2c00 / $2e00
        010             COLORE  64 - COLORE  95         $4c00 / $4e00
        011             COLORE  96 - COLORE 127         $6c00 / $6e00
        100             COLORE 128 - COLORE 159         $8c00 / $8e00
        101             COLORE 160 - COLORE 191         $ac00 / $ae00
        110             COLORE 192 - COLORE 223         $cc00 / $ce00
        111             COLORE 224 - COLORE 255         $ec00 / $ee00
```

Questa tabella esplica come riusare i vecchi registri colore da *$180* a *$1be* per accedere ai 256 colori. A destra sono riportati i valori che devono assumere i bit 13, 14, 15 del *$dff106* (BPLCON3) per accedere ai vari banchi. Facciamo un esempio: Se si vuol cambiare il colore 33, occorre fare:

```
1       DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI
2       dc.w    $182,$47f       ; metto i nibble alti
3       DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI
4       dc.w    $182,$3ea       ; metto i nibble bassi
```

Infatti si deve scegliere il banco che va dal colore 32 al 63, e di conseguenza scrivere nel *$dff180* significherà scrivere nel colore 32; scrivere nel *$dff182* significherà scrivere nel colore 33, e così via, fino al *$dff1be*, che normalmente sarebbe il colore 31, ma che in questo caso diventa il colore 63, ossia 31+32. Se avessimo scelto il banco che va dal colore 64 al 95 il *$dff182* sarebbe stato il colore 65, eccetera.

Ecco la lista dei valori per *$dff106* pronti da mettere in copperlist, può esservi utile per operazioni di "taglia e incolla" con *<Amiga+b+c+i>*:

```
1       DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
2       DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
3       DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI
4       DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI
5       DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI
6       DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI
7       DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI
8       DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI
9       DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI
10      DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI
11      DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI
12      DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI
13      DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI
14      DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI
15      DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI
16      DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI
```

Tutto sembrerebbe perfetto. Ma manca un particolare! Come si fa a scegliere 8 bitplanes nel BPLCON0? C'è il posto solo per 7 bitplanes. Infatti, sono disponibili i bit 12,13 e 14, che possono andare da *%000* per zero bitplanes a *%111* per 7 bitplanes, ossia 128 colori. Occorrerebbe poter avere un bit alto in più, in modo da ottenere *%1000*, ossia 8. Nessun problema, tale bit è stato assegnato come il quarto bit di *$dff100*. Per settare 8 bitplanes, dunque, occorre azzerare i bit 12,13,14 e settare il quarto, e il gioco è fatto. Ad esempio:

```
                    ;5432109876543210
        dc.w    $100,%0000001000010001  ; 8 bitplanes lowres (320*256)
        dc.w    $100,%1000001000010001  ; 8 bitplanes hires (640*256)
        dc.w    $100,%0111001000000001  ; 7 bitplanes lowres (320*256)
```

Da notare che lasciamo sempre il bit 9 settato, per il genlock, e settiamo il bit 0, ECSENA, che abilita dei bit speciali che vedremo in seguito. Da notare che si possono avere anche 6 bitplanes non extra half bright, cioè 64 colori di cui si può scegliere la palette normalmente, basta scegliere 6 bitplanes e settare il bit 9 (KillEHB) del BPLCON2 (*$dff104*). Se questo bit non è settato viene emulato il vecchio EHB, con 32 colori + 32 "scuriti". Ora, per verificare quanto abbiamo detto, apprestiamoci a visualizzare una figura a 256 colori, in Lezione15c.s

La pic è mia. Ammetto di essermi ispirato allo stile di giochi come *AGONY* e *SHADOW OF THE BEAST*, niente di artisticamente innovativo, ma mi pare che regga, no? Comunque serve bene allo scopo del listato. Avrete notato che prima della copperlist e della figura ci sono dei:

```
CNOP    0,8     ; allineo a 64 bit
```

In realtà con FMODE (*$dff1fc*) azzerato non "serve", vedrete dopo perché. Come abbiamo visto per le figure non aga, si può "attaccare" la palette in fondo alla figura, da mettere in copperlist con una routine. Tale routine è un pò più complessa, ma non poi troppo: `Lezione15c2.s`

Ora che abbiamo quella routine, vi sarà più facile capire come ottenere un fade a 24 bit, modificando la routine di fade vista nella lezione 8. Vedetevela in `Lezione15c3.s`. Ora vediamo di "ottimizzarla", in `Lezione15c4.s`. Infine la facciamo in "realtime" al 100%: `Lezione15c5.s`.

Ora potete provare a convertire la vostra figura in 320*256 a 128 o 256 colori, come preferite, usate il PicCon, l'IffConv o l'AgaConv nel disco di utility. Vi consiglio vivamente di leggere le istruzioni del PicCon presenti nel disco.

15.3 FMode

Siete riusciti a visualizzare la vostra figura AGA? Ebbene, se provaste a visualizzare una figura in hires 640*256, nonostante includiate il RAW e la PALETTE giusta e settiate il bit 15 del BPLCON0, non otterreste altro che uno schermo nero... Questo perché abbiamo lasciato *$dff1fc* (FMODE) azzerato. Tale registro controlla il BURST, ossia il modo con cui vengono trasferiti i dati dalla memoria al "video".

Normalmente il trasferimento è a 16 bit, ma per visualizzare la grafica più "spinta" occorre settare il trasferimento a 32 bit o a 64 bit. Se il trasferimento avviene a 16 bit, ciò che deve essere trasferito deve essere ad un indirizzo pari, ossia allineato a WORD (16 bit). Infatti non si deve puntare un bitplane ad un indirizzo dispari! Ora, se il burst è a blocchi di 32 bit, i dati devono essere ad un indirizzo allineato a 32 bit, ossia a longword! Per esempio un indirizzo come *$16dfc* è un multiplo di 4 (4*23423) e come tale multiplo di 4 bytes da 8 bit ossia di 4*8=32 bit. Insomma è un indirizzo allineato a 32 bit.

Per allineare dei dati ad indirizzi a 32 bit esiste la direttiva `CNOP 0,4`. Mentre EVEN, ossia `CNOP 0,2`, allinea a 2 byte, ossia 16 bit, cnoppare 0,4 allinea a 4 bytes, ossia 32 bit.

Se il burst è a 64 bit occorre mettere dei `CNOP 0,8` prima delle copperlist, degli sprites e dei bitplanes, per assicurarci l'allineamento a 64 bit. Se l'assemblatore facesse delle storie e non allineasse, la figura apparirebbe come "affettata", ossia a strisce verticali, dato che i blocchi a 32 o 64 bit non corrispondono all'inizio della figura. Per verificare se una label è allineata a 64 bit, assemblate, poi verificate a che indirizzo si trova tale label col comando "M", poi dividete l'indirizzo per 8, e moltiplicate di nuovo il risultato per 8. Se torna l'indirizzo originario, significa che è un multiplo di 8, e tutto è OK, se viene diverso significa che c'è un resto e non è un multiplo di 8. Allora mettete dei `dc.w 0` sopra l'indirizzo e provate ad allinearlo "a mano".

Naturalmente, sarebbe bene tenere sempre attivato il burst (bandwidth) al massimo, ossia a 64 bit. Questo si può fare mettendo il valore 3 nel *$dff1fc*. Comunque dovete stare attenti al fatto che se volete allargare i bitplanes, devono essere allargati a "blocchi" di 8 bytes alla volta. Per esempio abbiamo visto come convenga in certi casi avere "a lato" un pezzo di bitplane fuori dalla finestra video, per esempio per gli scroll e per i textscroll col blitter. In questo caso non potremmo aggiungere 2 bytes solamente, ma 8. Altro fatto è che non si deve usare MAI l'Allocmem per trovare spazio in memoria per i bitplanes, perché da degli indirizzi allineati a 16 bit, che solo per caso possono essere allineati anche a 64 bit. Già nei primi sorgenti della lezione, anche se non era necessario, è stata seguita la regola dell'allineamento:

```
1       CNOP  0,8         ; allineo ad indirizzo a 64 bit
2  sprite:
3         incbin "agasprite1"
4
5       CNOP  0,8         ; allineo ad indirizzo a 64 bit
6  pic:
7         incbin "AGAbitplanes"
```

Vediamo un pò meglio i primi due bit del registro FMODE (*$dff1fc*):

```
bit 1   BPAGEM  | Bitplane Modo pagina (doppio CAS)
bit 0   BLP32   | Bitplane largo 32 bit
```

Abbiamo detto che, se i bit sono entrambi azzerati, il burst è "emulazione OCS/ECS", ossia il trasferimento è a 16 bit. E se sono tutti e due settati il modo è 64bit. Vediamo i 4 casi in cui si possono trovare i primi 2 bit:

```
[x1]    %00     - Trasferimento bitplane data di 2 bytes alla volta (16bit)
                  Cicli di memoria: CAS normale
                  Larghezza del bus 16 bit
                  Richiesto: Bitplanes allineati a 16 bit

[x2]    %01     - Trasferimento bitplane data di 4 bytes alla volta (32bit)
                  Cicli di memoria: CAS normale
                  Larghezza del bus 32 bit
                  Richiesto: Bitplanes allineati a 32 bit (Double)
                             Modulo = Modulo -4

[x2]    %10     - Trasferimento bitplane data di 4 bytes alla volta (32bit)
                  Cicli di memoria: CAS DOPPIO
                  Larghezza del bus 16 bit
                  Richiesto: Bitplanes allineati a 32 bit (Double)
                             Modulo = Modulo -4

[x4]    %11     - Trasferimento bitplane data di 8 bytes alla volta (64bit)
                  Cicli di memoria: CAS DOPPIO
                  Larghezza del bus 32 bit
                  Richiesto: Bitplanes allineati a 64 bit (Quadruple)
                             Modulo = Modulo -8
```

Direi che va benissimo usare sempre *%11*, ossia *$3*. l'unico problema che può presentarsi è un aggrovigliamento dei DMA se eventualmente il blitter e il processore (non dotato di FAST RAM) dovessero inciampare nel fiume a 64bit del trasferimento ipergalattico dei chip AGA. In caso di queste turbolenze, potreste optare per un *%01* o *%10*, se vedete miglioramenti. Vediamo ora la bandwidth minima necessaria per le varie risoluzioni grafiche AGA (anche se tenteremo sempre di metterla a 64bit!).

Come già visto, per il 320*256 lowres a 8 bitplanes bastano 16bit (*$1fc,0*):

```
LORES (320x256),        Per 64, 128 ,256 colori o HAM8, bastano 16bit

HIRES (640x245),        Per 32, 64, 128, 256 colori o HAM8, occorrono 32bit

SUPERHIRES (1280x200)   Per 2, 4 colori bastano 16 bit
                        Per 8, 16 colori occorrono 32 bit
                        Per 32, 64, 128, 256, HAM8 occorrono 64 bit
```

Intanto, potremmo cominciare mettendo il BURST al massimo nella visualizzazione della nostra tranquilla figura in lowres. Anche se visibilmente non accadrà nulla, il trasferimento sarà più GALATTICO. Occorre però una precisazione IMPORTANTISSIMA: Cambiare il FETCH comporta anche una correzione del MODULO, per circostanze hardware. Dunque, se l'FMODE è azzerato, e il trasferimento avviene a 16 bit, il modulo deve essere 0, o comunque è normale. Se invece il BURST è a 32 bit, il modulo è uguale al modulo -4, per cui se era zero, occorre mettere -4 in BPL1MOD/BPL2MOD per compensare. Se il BURST è a 64 bit, il modulo è uguale al modulo normale -8:

```
BANDWIDTH 1: dc.w $1FC,0        ; Allora i bitplanes devono essere allineati
                                ; almeno a word (16 bit), e il modulo è
                                ; quello normale.

BANDWIDTH 2: dc.w $1FC,1 o 2    ; Allora i bitplanes devono essere allineati
                                ; almeno a long (32 bit), e il modulo è
                                ; uguale al modulo normale meno 4

BANDWIDTH 4: dc.w $1FC,3        ; Allora i bitplanes devono essere allineati
                                ; almeno a quadword (64 bit), e il modulo è
                                ; uguale al modulo normale meno 8
```

Per verificare il tutto, ricaricate Lezione15c.s, e provate a modificare l'FMODE in copperlist mettendoci il valore 1 o 2, attivando il burst a 32 bit. Noterete che se non fate altre modifiche la figura appare con il modulo sbagliato. Modificate allora anche il modulo, mettendolo = -4, e vedrete che la pic si "riaddrizza". Allo stesso modo, provate a mettere il burst a 64 bit, mettendo il valore *$3* al `dc.w $1fc` (FMODE) in copperlist. Ora dovrete mettere il modulo, sia *$108* che *$10a*, a -8 per vedere la PIC.

Chiarito questo fatto, tenete sempre l'FMODE a *$3*, ossia settate sempre i primi 2 bit, e potrete visualizzare anche hires a 256 colori.

C'è un'ultimo particolare riguardo agli effetti del burst a 32 o 64 bit. Anche i valori del DDFSTRT e DDFSTOP sono modificati. Con un burst normale a 16 bit per aprire uno schermo in hires che partisse alla posizione orizzontale MIOX, si determinava con la "formula":

```
DDFSTRT=(MIOX-9)/2
```

Con il burst a 32 bit, invece, occorre fare:

```
DDFSTRT=(MIOX-17)/2
```

Perchè viene letta una longword anzichè una word. Comunque se usate schermi a larghezza standard non ci sono problemi, e se ci fossero potete andare a tentativi! Comunque, in pratica, con il burst attivo, se visualizzate una figura in hires non occorre settare il DDFSTART e il DDFSTOP a *$003c* e *$00d4*, ma allo stesso modo del lowres:

```
1       dc.w    $92,$0038       ; DdfStart lowres, adatto per HIRES con burst
2       dc.w    $94,$00d0       ; DdfStop lowres, come sopra
```

Questo a causa dei cicli di memoria richiesti per un trasferimento “turbo” dalla ChipRam al Chip Lisa.

Vediamo di visualizzare una figura in hires a 256 colori in `Lezione15d.s` La figura in questione è opera di Cristiano Evangelisti, handle *KREEX*, un “indipendente” che sta facendo la grafica ad un gioco adventure che sta programmando un mio allievo.

15.4 HAM8

Il buon vecchio HAM a 6 bitplanes è stato “sbancato” dal nuovo HAM8, a 8 bitplanes. 6 bitplanes sono usati per i colori e 2 per i bit di controllo. Inoltre è disponibile in tutte le risoluzioni, non solo in LowRes. Per attivarlo, basta settare 8 bitplanes e il bit dell'HAM nel `BPLCON0` (*$100*). Degli 8 bit, i 6 bit alti sono usati come 64 registri colore base a 24bit, o come un valore di MODIFY a 6 bit, più i 2 bit bassi per il modo hold o modify a 18bit. Questo permette di visualizzare più di 256000 colori. I 2 planes di controllo e i 6 piani colore sono “internamente” fusi negli 8 bit dell'HAM8, rovesciando però l'ordine: prima i piani 3, 4, 5, 6, 7, 8 poi 1 e 2. Questo causa degli scambi di bitplanes che vedremo.

Ecco un confronto tra il vecchio HAM6 e il nuovo HAM8.

Funzione dei bitplanes di controllo 5 e 6 nell'HAM6:

```
+-----+-----+--------+--------+------------------+
| BP6 | BP5 |   RED  |  GREEN | BLUE             |
+-----+-----+--------+--------+------------------+
| 0   | 0   | selez.il nuovo reg.base (1 dei 16) |
+-----+-----+--------+--------+------------------+
| 0   | 1   |  hold  |  hold  | modify           |
+-----+-----+--------+--------+------------------+
| 1   | 0   | modify |  hold  |  hold            |
+-----+-----+--------+--------+------------------+
| 1   | 1   |  hold  | modify |  hold            |
+-----+-----+--------+--------+------------------+
```

Nell'HAM8 i bitplane di controllo sono l'1 e il 2:

```
+-----+-----+--------+--------+------------------+
| BP2 | BP1 |   RED  |  GREEN | BLUE             |
+-----+-----+--------+--------+------------------+
| 0   | 0   | selez.il nuovo reg.base (1 dei 64) |
+-----+-----+--------+--------+------------------+
| 0   | 1   |  hold  |  hold  | modify           |
+-----+-----+--------+--------+------------------+
| 1   | 0   | modify |  hold  |  hold            |
+-----+-----+--------+--------+------------------+
| 1   | 1   |  hold  | modify |  hold            |
+-----+-----+--------+--------+------------------+
```

Questi 2 bit BASSI sono il comando: nuovo registro di base o altera una delle componenti RED, GREEN, BLU. Attenzione al fatto che i 2 bit bassi del colore non possono essere modificati, per cui la palette iniziale deve essere scelta bene. (Comunque questo spetta ai grafici e ai programmi come AdPro o ImageFX).

Ora, vediamo in pratica come visualizzare una figura HAM8. Innanzitutto la palette è di 64 colori, non di 256: bastano infatti solo quei “pochi” colori per generare l'ham, grazie ai bit di controllo che “holdano” o “modificano” le componenti RGB. Per attivarlo basta settare in `BPLCON0` 8 bitplanes e il modo HAM, ossia settare il bit 4 e il bit 11. C'è però un'ultima “particolarità”. Abbiamo già detto che i bitplanes 1-2 sono internamente “scambiati” con i bitplanes 3-4-5-6-7-8.

Ebbene, in effetti al momento di "puntare" i bitplanes c'è questo problema. Se salvate il RAW con il PicCon potete puntare regolarmente la figura, come avete fatto per una figura a 256 colori. Questo perchè il PicCon scambia già l'ordine nel RAW, in modo che poi "torni". Se invece salvate il raw con l'AgaConv o con altri iffconverters, il raw sarà salvato "come è", per cui dovrete far puntare i primi 6 bitplanes come fossero i bitplanes 3,4,5,6,7,8, e infine puntare i plane 1 e 2.

```
1  ; Questo è l'ordine dei bitplanes se salvate il RAW con AgaConv o con un
2  ; altro iffconverter che non "rovescia" i plane da solo.
3
4  BPLPOINTERS:
5          dc.w $e8,0,$ea,0        ; terzo         bitplane
6          dc.w $ec,0,$ee,0        ; quarto            "
7          dc.w $f0,0,$f2,0        ; quinto            "
8          dc.w $f4,0,$f6,0        ; sesto             "
9          dc.w $f8,0,$fA,0        ; settimo           "
10         dc.w $fC,0,$fE,0        ; ottavo            "
11         dc.w $e0,0,$e2,0        ; primo             "
12         dc.w $e4,0,$e6,0        ; secondo           "
```

Nel listato di esempio il RAW è salvato con PicCon, per cui i planes sono puntati in modo normale. Caricate `Lezione15e.s`

Ora possiamo fare un confronto tra l'HAM8 e i normali 256 colori. Vedete e giudicate in `Lezione15e2.s`

Da notare che cambiare l'intera palette AGA è un'operazione che richiede una decina di linee raster! In questo esempio cambiamo "solo" 64 colori, per cui bastano 2 o 3 linee, ma se volessimo fare, ad esempio, un gioco adventure con una pic a 256 colori nella parte alta dello schermo, e un pannello di controllo nella parte bassa, al momento del cambio della palette dovremmo lasciare 10 pixel "neri", in attesa che la palette cambi del tutto, comunque occorre considerare il tempo di "repalettamento". Vi ricordate che ogni `MOVE` del copper impiega 8 pixel lowres, e circa 52 move occupano una linea?...

15.5 Sprite

Le novità per quanto riguarda gli sprites sono molteplici. Come prima cosa è possibile deciderne la larghezza, scegliendo tra 16, 32 o 64 pixel. Come sapete normalmente la larghezza massima era di 16 pixel! Inoltre lo sprite può essere visualizzato in lowres, hires o superhires, indipendentemente dalla risoluzione della pic sullo sfondo. Vediamo come si fanno, in pratica, queste cose.

La risoluzione dello sprite si decide con i bit 6 e 7 del registro `BPLCON3` (*$dff106*), ed è indifferente la larghezza degli sprite:

```
bit 7   bit 6

  0       0       LOW RES, emulazione OCS/ECS (140ns)
  0       1       LOW RES, (140ns) (Non è il modo ECS standard!)
  1       0       HIRES (70ns)
  1       1       SUPER HIRES (35ns)
```

Questi due bit si chiamano `SPRES0` e `SPRES1`, tanto per cambiare. Vediamo subito un esempio di sprite in hires, dato che basta settare il bit 7 del *$dff106*, in `Lezione15f.s`

Sprites larghi 32 o 64 pixel

Ora occorre vedere come è possibile fare sprites larghi 32 o 64 pixel. Innanzitutto è necessario avere un iffconverter che salvi sprites di questo tipo! Il PicCon o AgaConv li salvano adeguatamente, non ci sono problemi. Come al solito ci sono un paio di bit che decretano la larghezza.

Questi sono i bit 3 e 2 del registro FMODE (*$dff1fc*), detti SPAGEM e SPR32. I bit SPAGEM e SPR32 decidono la larghezza dello sprite, e di conseguenza se i dati da passare a SPRxDATA devono essere a 16, 32 o 64 bit, in modo analogo a come viene fatto per i bitplanes. è analogo anche il fatto che gli sprite a 32 bit devono essere allineati con un `cnop 0,4`, e quelli a 64 bit con un `cnop 0,8`. Questo per il risaputo fatto che i traferimenti a 16 bit richiedono una bandwidth *1, mentre quelli a 32 bit ne richiedono una *2, di conseguenza quelli a 64 ne richiedono una *4. Nel caso degli sprite però variano le word di controllo, che si "allargano" assieme al resto dello sprite, nei casi sia a 32 o 64 bit.

Ma vediamo una tabella dei valori dei bit SPAGEM e SPR32 dell'FMODE:

```
bit 3 | bit 2 | Larghezza   | Word di controllo
------+-------+-------------+---------------------------------
  0   |   0   | 16 pixel    | 2 word (normale) - richiede cnop 0,2
  1   |   0   | 32 pixel    | 2 longword - richiede cnop 0,4
  0   |   1   | 32 pixel    | 2 longword - richiede cnop 0,4
  1   |   1   | 64 pixel    | 4 longword - richiede cnop 0,8
-----------------------------------------------------------------
```

Gli sprite "allargati" non sono disponobili nel caso in cui il DMA non ce la faccia, specialmente in superhires overscan interlacciato a 256 colori.

Dunque, avendo salvato uno sprite largo 32 o 64 pixel con l'iffconverter, e avendolo allineato ad un indirizzo multiplo di 4 o di 8, possiamo accedere alle sue word di controllo nello stesso modo di uno sprite largo 16 pixel? NO di certo, ecco perché:

Questa è la struttura di uno sprite normale, largo 16 pixel:

```
1  MIOSPRITE16:
2  VSTART:
3          dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
4  HSTART:
5          dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
6  VSTOP:
7          dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
8  VHBITS:
9          dc.b $00        ; bit
10
11  dc.w   %0000000000000000,%0000110000110000 ; dati
12  dc.w   %0000000000000000,%0000111001110000
13  ...
14  dc.w   0,0     ; 2 word azzerate definiscono la fine dello sprite.
```

Ossia:

```
1  ---------------------------------------------------------------------
2  dc.w 0,0                        ; 2 word di controllo
3  dc.w dataPlane1,dataPlane2      ; 2 word (16 bit - 16 pixel) con i 2 "plane"
4  dc.w dataPlane1,dataPlane2      ; 2 word (16 bit - 16 pixel) con i 2 "plane"
5  dc.w dataPlane1,dataPlane2      ; 2 word (16 bit - 16 pixel) con i 2 "plane"
6  ....
7  dc.w 0,0                        ; 2 word azzerate per terminare
8  ---------------------------------------------------------------------
```

Ora, la struttura degli sprite larghi 32 pixel è questa:

```
1  ---------------------------------------------------------------------
2  dc.l 0,0                        ; 2 longword di controllo
3  dc.l dataPlane1,dataPlane2      ; 2 longword (32 bit/pixel) con i 2 "plane"
4  dc.l dataPlane1,dataPlane2      ; 2 longword (32 bit/pixel) con i 2 "plane"
5  dc.l dataPlane1,dataPlane2      ; 2 longword (32 bit/pixel) con i 2 "plane"
6  ....
7  dc.l 0,0                        ; 2 longword azzerate per terminare
8  ---------------------------------------------------------------------
```

Mentre quella degli sprite larghi 64 pixel è questa:

```
 dc.l 0,0,0,0                           ; 2 doppie longword di controllo
 dc.l data1a,data1b,data2a,data2b       ; 2 doppie longword (64 bit/pixel)
 dc.l data1a,data1b,data2a,data2b       ; 2 doppie longword (64 bit/pixel)
 dc.l data1a,data1b,data2a,data2b       ; 2 doppie longword (64 bit/pixel)
 ....
 dc.l 0,0,0,0                           ; 2 doppie longword = 0 per terminare
```

Ora, quello che ci interessa è trovare i nostri bit nelle nuove word di controllo estese a longword e a doppia longword. Per quanto riguarda gli sprite larghi 32 bit:

```
SPRITE32:
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        DC.W 0          ; Word "aggiunta" nello sprite largo 32 pixel
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        DC.W 0          ; Word "aggiunta" nello sprite largo 32 pixel

 dc.l %00000000000000011110000000000000,%00000000000010000000000000000000 ; dati
 dc.l %00000000000011111111000000000000,%00000000000101111100000000000000
 ...
 dc.l   0,0             ; Fine dello sprite (2 longword anzichè 2 word).
```

Come si vede, le 2 word di controllo sono diventate 2 long, e i bit di controllo sono rimasti quelli della word alta.

Vediamo ora il caso dei pixel larghi 64 pixel:

```
SPRITE64:
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)

 dc.l data1a,data1b,data2a,data2b       ; 2 doppie longword (64 bit/pixel)
 dc.l data1a,data1b,data2a,data2b       ; 2 doppie longword (64 bit/pixel)
 dc.l data1a,data1b,data2a,data2b       ; 2 doppie longword (64 bit/pixel)
 ...
 dc.l   0,0,0,0         ; Fine dello sprite (2 doppie longword!).
```

Da qui ne deriva che occorre fare delle piccole modifiche a UniMuoviSprite, in modo che acceda ai byte spostati della seconda word di controllo.

Un esempio di sprite largo 32 pixel è `Lezione15f2.s`

Un esempio di sprite largo 64 pixel è `Lezione15f3.s`

Avete visto che insettone? E ne potete fare 8 così, oppure 4 a 16 colori in modo attacched.

Da notare che il PicCon salva gli sprite attacched senza il bit 7 (attach) settato allo sprite dispari, per cui lo dovete settare “a mano” se lo salvate con questo IffConverter.

15.6 Nuovo posizionamento orizzontale ad 1/4 di pixel

Un quarto di pixel? Ebbene si! Sono stati aggiunti 2 bit "bassi" alla posizione orizzontale dello sprite, rendendola possibile a "scattini" 4 volte più piccoli, dunque più fluidi. Vediamo dove sono stati messi questi bit, analizzando SPRxCTL, la serie di registri che è un "equivalente" della seconda word di controllo dello sprite:

```
 $dff142/14A/152/15A/162/16A/172/17A - SPRxCTL - Controllo e posiz. sprite

+-------+-------+-----------------------------------------------------+
| BIT   |  nome |       FUNZIONE                                      |
+-------+-------+-----------------------------------------------------+
| 15-08 | EV7-0 | VSTOP - Gli 8 bit bassi della posiz. vert. di fine)   |
| 07    | ATT   | Bit di controllo dell'attached (solo sprite dispari)  |
| 06    | SV9   | Decimo bit della posizione di inizio verticale        |
| 05    | EV9   | Decimo bit della posizione di fine verticale          |
| 04    | SH1=0 | Posiz. orizzontale, incremento 70nS (mezzo pixel)     |
| 03    | SH0=0 | Posiz. orizzontale, incremento 35nS (1/4 di pixel)    |
| 02    | SV8   | Nono bit della posizione verticale di inizio (vstart) |
| 01    | EV8   | Nono bit della posizione verticale di fine (vstop)    |
| 00    | SH2   | Posiz. orizzontale, incremento 140nS (1 pixel)        |
+-------+-------+-----------------------------------------------------+
```

I bit che ci interessano sono SH0, SH1, SH2, ossia Start Horizontal. Come vedete, oltre ai noti bit SV8, EV8, ossia gli ottavi bit di VSTART e VSTOP, troviamo anche due nuovi bit che riguardano l'HSTART: oltre al bit "basso" che ci permette lo scroll di un pixel per volta, ne sono stati aggiunti un paio di ancora più "bassi", che ci permettono di fare scroll di mezzo pixel o di 1/4 di pixel alla volta. Per 140ns (nanosecondi) si intende il "tempo" video di scorrimento. è comunque più chiaro dire che 140ns corrispondono ad 1 pixel lowres, mentre 140/2= 70ns corrispondono ad un pixel hires, (o mezzo pixel lowres), infine appare ovvio che 70/2= 35ns equivalgono ad 1/4 di pixel lowres, oppure ad un pixel in risoluzione 1280*xxx, ossia superhires.

Ma come si fa allora a far spostare a scattini di 1/4 di pixel lo sprite? Una via è quella di modificare la routine UniMuoviSprite, in modo che in entrata accetti una posizione X da 0 a 1280, anziché da 0 a 320, in questo modo se aggiungiamo 1 ogni volta lo scroll sarà ad 1/4 di pixel, se aggiungiamo 2 sarà a mezzo pixel oppure aggiungendo 4 per volta avremo uno scroll di un pixel alla volta. Semplice, no?

Vedetevi l'implementazione in Lezione15f4.s e Lezione15f5.s

In pratica la posizione orizzontale AGA è un numero a 11 bits, anziché 9.

15.7 Il bit BRDRSPRT

Il bit BRDRSPRT, quando è settato, rende possibile la visualizzazione degli sprite anche fuori dai bordi definiti da DIWSTART/DIWSTOP. Da notare che con questo bit abilitato gli sprite sono visualizzati anche se i bitplanes sono disabilitati in BPLCON0! Occorre però ricordarsi di settare anche il bit 0 del bplcon0 ($dff100), che abilita anche altri bit speciali. Il bit in questione è il secondo (01) del $dff106 (BPLCON3).

Vediamo una sua implementazione in Lezione15f6.s

15.8 Il modo attached

Gli sprites possono essere attacched in qualsiasi modo, escluso il modo ECS SHRES (1280*xxx, 35ns).

15.9 La palette degli sprite AGA

Si può usare come palette per gli sprite un qualsiasi banco di 16 colori preso dalla palette di 256. I bit da ESPRM7 a ESPRM4 permettono di "rilocare" la colomap degli sprite pari, mentre i bit da OSPRM7 a OSPRN4 permettono di rilocare la colormap degli sprite dispari. Negli OCS/ECS i 16 colori degli sprite erano sempre e obbligatoriamente dal color16 (*$dff1a0*) al color31 (*$dff1be*), per cui una figura che aveva più di 16 colori doveva per forza condividere i colori dal 16 al 31 con gli sprite. Con l'AGA invece è possibile spostare questo banco di 16 colori in un qualsiasi segmento dei 256. Se per esempio avessimo una figura a 128 colori, potremmo spostare i colori degli sprite alla posizione dal color129 in avanti, in modo da non dover condividere la palette con la figura. L'utilità quindi si riscontra con le figure con più di 16 colori. Comunque se i bitplanes sono 8 e i colori 256, possiamo scegliere quale banco di 16 usare, ma tali 16 colori saranno sempre in comune con la figura. Ecco come sono assegnati nell'OCS i colori della palette agli sprite:

```
Sprites | Colors
-----------------
   0-1  |  00-03        ; $dff1a0/1a2/1a4/1a6
   2-3  |  04-07
   4-5  |  08-11
   6-7  |  12-15
-----------------
```

Quindi ci sono 4 coppie di sprite a 3 colori. Per esempio, per definire i 3 colori del primo sprite:

```
1         dc.w    $1A2,$462       ; color17, nibble bassi
2         dc.w    $1A4,$2e4       ; color18, nibble bassi
3         dc.w    $1A6,$672       ; color19, nibble bassi
```

Nel chipset AGA, invece, oltre a poter scegliere quale parte della palette a 256 colori usare per gli sprite, è possibile selezionare 2 palette, una per gli sprite pari e una per gli sprite dispari, avendo un totale di 32-8 colori, ossia 24, dato che il color0 è il trasparente e non conta. Ricapitolando, mentre in OCS avevamo 8 sprite a 3 colori effettivi ciascuno, ma legati da un rapporto di "coppia", i colori totali erano 3*4=12, nell'AGA gli sprite sono sempre a 3 colori, ma non condividono la palette a coppie! Comunque, se gli sprite sono attacched, viene usata per tutti la stessa palette da 16 colori, quella assegnata agli sprite dispari.

Dunque, nella palette AGA a 256 colori, abbiamo 16 palette da 16 colori tra cui scegliere, usando il byte basso del BPLCON4 ($dff10c). I bit dal 7 al 4 sono usati per scegliere il "numero" della sottopalette di 16 da usare per gli sprite pari, mentre i bit dal 3 allo 0 sono per scegliere la sottopalette degli sprite dispari.

Vediamo gli 8 bit bassi del registro BPLCON4 (*$dff10c*):

```
bit      "nome"

 0       ESPRM7 \   Scegli la sottopalette da
 1       ESPRM6  \  usare per gli sprite PARI
 2       ESPRM5  /
 3       ESPRM4 /
 4       OSPRM7 \   Scegli la sottopalette da
 5       OSPRM6  \  usare per gli sprite DISPARI
 6       OSPRM5  /
 7       OSPRM4 /
```

Ed ecco una tabella di riferimento per scegliere la palette:

```
bit 3 | bit 2 | bit 1 | bit 0 | Sprites pari
bit 7 | bit 6 | bit 5 | bit 4 | Sprites dispari
------+-------+-------+-------+----------------------------------------
  0   |   0   |   0   |   0   | $180/palette 0 (colore 0)
  0   |   0   |   0   |   1   | $1A0/palette 0 (colore 16)
  0   |   0   |   1   |   0   | $180/palette 1 (colore 32)
  0   |   0   |   1   |   1   | $1A0/palette 1 (colore 48)
  0   |   1   |   0   |   0   | $180/palette 2 (colore 64)
  0   |   1   |   0   |   1   | $1A0/palette 2 (colore 80)
  0   |   1   |   1   |   0   | $180/palette 3 (colore 96)
  0   |   1   |   1   |   1   | $1A0/palette 3 (colore 112)
  1   |   0   |   0   |   0   | $180/palette 4 (colore 128)
  1   |   0   |   0   |   1   | $1A0/palette 4 (colore 144)
  1   |   0   |   1   |   0   | $180/palette 5 (colore 160)
  1   |   0   |   1   |   1   | $1A0/palette 5 (colore 176)
  1   |   1   |   0   |   0   | $180/palette 6 (colore 192)
  1   |   1   |   0   |   1   | $1A0/palette 6 (colore 208)
  1   |   1   |   1   |   0   | $180/palette 7 (colore 224)
  1   |   1   |   1   |   1   | $1A0/palette 7 (colore 240)
-------------------------------------------------------------------------
```

Ecco come si usa: se per esempio volessi scegliere sia per gli sprite pari che per quelli dispari la seconda palette, quella dal color16 al color31, dovrei mettere *%0001* nei bit dallo 0 al 3 per gli sprite pari, e *%0001* nei bit dal 4 al 7 per gli sprite dispari. Per cui il byte basso sarebbe *%00010001*. Ora, questa mia preferenza corrisponde con la modalità dell'OCS/ECS, per la quale la palette degli sprite è sempre dal color16 al color31. Infatti, *%00010001* in esadecimale è *$11*, ed è per questo che facciamo:

```
1       move.w  #$11,$10c(a5)           ; BPLCON4 resettato
```

Per resettare la palette degli sprite!!! Svelato questo mistero, vediamo di cambiare il settaggio in un modo più utile per eventuali usi: spostiamo la palette degli sprite in fondo, ossia decidiamo che sia dal color240 al color256. In questo caso abbiamo *%11111111*. Ora potremmo però scegliere un banco da 16 per gli sprite pari diverso da quello per gli sprite dispari! Per esempio, assegnamo agli sprite pari i colori dal 224 al 240, e agli sprite dispari dal 240 al 256. Il risultato in *$dff10c* è *%11101111*.

Mettiamo in pratica questo in `Lezione15f7.s`

15.10 Nuovo scroll orizzontale superfluido (1/4 di pixel) per i bitplanes

Lo scrolling orizzontale ad 1/4 di pixel è stato implementato anche per i bitplanes. E indovinate come? Aggiungendo dei bit al `BPLCON1` (*$dff102*). Come si è visto per gli sprites, sono stati aggiunti un paio di bit "bassi" del valore di scroll. In più ne sono stati aggiunti anche due alti, che permettono scatti di 16 pixel alla volta, per un massimo di 64 pixel. Da notare che gli scatti di 16 e 32 pixel sono "abilitati" solo quando il burst (`FMODE`-*$dff1fc*) è a 32 o 64 bit, rispettivamente. Dunque ora lo scroll può andare da 0 a 64 pixel, a scatti di 1/4 di pixel. Ma recapitoliamo: Se prima il valore di scostamento orizzontale di ciascun playfield poteva andare da 0 a 15 (*%1111*), e sono stati aggiunti 2 bit bassi e due alti, ora può andare da 0 a *%11111111*, ossia da 0 a 255 (un valore a 8 bit!), da intendersi però come scatti da 1/4 di pixel, per cui lo scroll massimo se misurato in pixel lowres è di 256/4=64. Ma vediamo in che posizione questi bit sono stati "incastrati" nel byte alto del vecchio bplcon1 (*$dff102*):

```
        BIT     "nome"          descrizione

        15      PF2H7   \ bit alti (6 e 7) del valore scroll playfield 2
```

```
        14      PF2H6   /
        13      PF2H1   \ bit bassi (0 e 1) del valore scroll playfield 2
        12      PF2H0   /
        11      PF1H7   \ bit alti (6 e 7) del valore scroll playfield 1
        10      PF1H6   /
        09      PF1H1   \ bit bassi (0 e 1) del valore scroll playfield 1
        08      PF1H0   /

        07      PF2H5   \
        06      PF2H4    \ bit "medi" (2,3,4,5) del val. scroll playfield 2
        05      PF2H3    /
        04      PF2H2   /
        03      PF1H5   \
        02      PF1H4    \ bit "medi" (2,3,4,5) del val. scroll playfield 1
        01      PF1H3    /
        00      PF1H2   /

Nota:
        Il bit PFxH0 scrolla di 1/4 di pixel (35ns)
        Il bit PFxH1 scrolla di 1/2 pixel (70ns)
        Il bit PFxH2 scrolla di 1 pixel (140ns)
        Il bit PFxH3 scrolla di 2 di pixel
        Il bit PFxH4 scrolla di 4 di pixel
        Il bit PFxH5 scrolla di 8 di pixel
        Il bit PFxH6 scrolla di 16 pixel (deve essere attivo il burst 32 bit)
        Il bit PFxH7 scrolla di 32 pixel (deve essere attivo il burst 64 bit)
```

Come vedete il byte basso è lo stesso, mentre quello alto è AGA only.

Ora, supponiamo di voler far scorrere un bitplane verso destra a scattini di 1/4 di pixel, fino al massimo possibile con il bplcon1, ossia di 256 posizioni equivalenti a 64 pixel lowres. Dovremmo scomporre il valore di scoll, da 0 a 255, in 3 "pezzi": i due bit bassi dovrebbero essere messi in PHxH0/1, i 4 "centrali" in PFxH2-5, e i due bit alti in PFxH6/7. Questo si può fare facilmente con qualche AND e LSL/LSR.

Vediamo una implementazione in Lezione15g1.s (1 playfield)

Vediamo un'implementazione in Lezione15g2.s (2 playfield)

Ora proviamo a fare un'effetto "onda" con la precisione di 1/4 di pixel, conventendo una sintab in valori per BPLCON1 e cambiando quest'ultimo in copperlist una volta per linea: Lezione15g3.s

Una particolarità: se la figura è in hires, non "funziona" il bit più alto dello scroll, per cui i valori possono andare da 0 a 127. Lezione15g4.s

15.11 Una nuova possibilità per ciclare la palette

Abbiamo già visto la funzione dei bit bassi del BPLCON4. I bit alti, invece, servono per "scambiare" colori nella palette, senza dover cambiare il contenuto dei registri della palette stessa.

```
        BPLTCON4 ($dff10c)

       BIT      NOME

        15      BPLAM7
        14      BPLAM6
        13      BPLAM5
        12      BPLAM4
        11      BPLAM3
        10      BPLAM2
```

```
09      BPLAM1
08      BPLAM0
```

BPLAMx = This 8 bit field is XOR'ed with the 8 bit plane color address,thereby altering the color address sent to the color table (x=1-8) Bits 15 thru 8 of BPLCON4 comprise an 8-bit mask for the 8 bitplane address, XOR'ing the individual bits. This allows the copper to exchange colour maps with a single instruction.

Vediamo un esempio pratico di scambio tra il colore A e il colore B:

- Il contenuto del registro colore hardware non è modificato
- Tutti i pixel che venivano visualizzati usando il colore A ora sono visualizzati usando il colore B, e tutti i pixel che erano visualizzati col colore B è visualizzato col colore A. (in pratica: SCAMBIATI!)
- Il gruppo di 2^n colori dal colore 00 al colore $(2^n) - 1$ è scambiato col gruppo di 2^n colori dal colore 2^n al colore $2^n + (2^n) - 1$
- Il gruppo di 2^n colori dal colore $2 * 2^n$ al colore $2 * 2^n + (2^n) - 1$ è scambiato con il gruppo di 2^n colori dal colore $3 * 2^n$ al colore $3 * 2^n + (2^n) - 1$

L'operazione di scambio finisce quando l'hardware non trova altri gruppi di colore da scambiare.

Facciamo un esempio: se settiamo il secondi bit, BPLAM1 (bit 9 del BPLCON4), ecco come appare la palette prima e dopo l'operazione:

```
   PRIMA          |       DOPO
-------------------------------------
   Color 00       |       Color 02
   Color 01       |       Color 03
   Color 02       |       Color 00
   Color 03       |       Color 01
   Color 04       |       Color 06
   Color 05       |       Color 07
   Color 06       |       Color 04
   Color 07       |       Color 05
   ...            |       ...
```

I colori sono stati scambiati, usando gruppi di $2^1 = 2$ colori.

Non si può scambiare un solo colore. Se si modifica un bit BPLAMx, si cambia tutta la palette.

Le operazioni di scambio comunque possono essere combinate. Se più di un bit BPLAMx è settato, le operazioni di scambio per ogni bit saranno eseguite una dopo l'altra, partendo dal bit BPLAM0 fino a BPLAM7.

Esempio: *$dff10c* contiene *$0500* (*%0000010100000000*). I bit BPLAM0 e BPLAM2 sono settati. Prima saranno scambiati usando gruppi di 2^0 colori, POI sarà scambiata la palette risultante usando gruppi di 2^2 colori come in questa tabella:

```
   PRIMA          |     Scambio BPLAM0    |     Scambio BPLAM1
-------------------------------------------------------------------
   Color 00       |       Color 01        |       Color 05
   Color 01       |       Color 00        |       Color 04
   Color 02       |       Color 03        |       Color 07
   Color 03       |       Color 02        |       Color 06
   Color 04       |       Color 05        |       Color 01
   Color 05       |       Color 04        |       Color 00
   Color 06       |       Color 07        |       Color 03
```

```
Color 07        |       Color 06        |       Color 02
Color 08        |       Color 09        |       Color 13
Color 09        |       Color 08        |       Color 12
Color 10        |       Color 11        |       Color 15
Color 11        |       Color 10        |       Color 14
Color 12        |       Color 13        |       Color 09
Color 13        |       Color 12        |       Color 08
Color 14        |       Color 15        |       Color 11
Color 15        |       Color 14        |       Color 10
...             |       ...             |       ...
```

In pratica gli 8 bit BPLAM0-7 del BPLCON4 sono una maschera per l'indirizzo degli 8 bitplanes, dato che viene fatto uno XOR (EOR) di ogni bit.

Vediamo un esempio, solo "didattico" degli effetti di questi bit: Lezione15h.s

15.12 Dual playfield AGA

Il nuovo Dual Playfield può avere fino a 4 bitplanes per playfield (16 colori un playfield e 16 l'altro), e il banco dei 16 colori nella palette di 256 è selezionabile indipendentemente per ogni playfield.

Per attivare il double Playfield, occorre settare il bit 10 del BPLCON0 come al solito, scegliere i bitplanes, (per 8 planes azzerare i bit 12, 13, 14 del BPLCON0 e settare il 4, altrimenti da 2 a 6 usare i bit 12, 13, 14 e azzerare il 4). Ora occorre puntare le 2 figure, una nei bplpointer pari e una nei bplpointers dispari. Poi si deve scegliere quali banchi di colori usare per le 2 pic, in mood simile a quanto visto per gli sprite.

Questo si decide con i bit 10, 11, 12 del BPLCON3 (*$dff106*):

```
| PF2OF     | BITPLANE COINVOLTI                | OFFSET     |
+---+---+---+---+---+---+---+---+---+---+---+---+------------+
| 2 | 1 | 0 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | (decimale) |
+---+---+---+---+---+---+---+---+---+---+---+---+------------+
| 0 | 0 | 0 | - | - | - | - | - | - | - | - | 0          |
| 0 | 0 | 1 | - | - | - | - | - | - | 1 | - | 2          |
| 0 | 1 | 0 | - | - | - | - | - | 1 | - | - | 4          |
| 0 | 1 | 1 | - | - | - | - | - | 1 | - | - | 8 (default)|
| 1 | 0 | 0 | - | - | - | 1 | - | - | - | - | 16         |
| 1 | 0 | 1 | - | - | 1 | - | - | - | - | - | 32         |
| 1 | 1 | 0 | - | 1 | - | - | - | - | - | - | 64         |
| 1 | 1 | 1 | 1 | - | - | - | - | - | - | - | 128        |
+---+---+---+---+---+---+---+---+---+---+---+---+------------+
```

Si intende Playfield 2 prioritario rispetto al Playfield 1. Come vedete la situazione di default avviene quando i bit 10 e 11 sono settati, infatti per default mettiamo *$c00* (*%110000000000*) in *$dff106*.

15.13 VGA/Productivity 640x480 non interlacciata

Le demo e i giochi funzionano normalmente in risoluzione PAL o NTSC, che sono supportate dai televisori o dai monitor come il 1084. La frequenza verticale pal è di 50Hz, mentre quella NTSC di 60Hz. La frequenza orizzontale è di 15Khz. Come sapete, per scegliere fra una di queste due frequenze si fa così:

```
1       move.w  #$20,$dff1dc    ; BEAMCON0 - modo PAL
2
3       move.w  #$00,$dff1dc    ; BEAMCON0 - modo NTSC
```

Questo però non funziona sui vecchi computer Amiga, fabbricati prima del 1990 o 1991. In pratica gli A1000 e i primi a500/a2000 non possiedono il FAT AGNUS, che "possiede" il BEAMCON0, mentre tale registro ha cominciato a comparire negli A500/a2000 kickstart 1.3 fabbricati dopo il 1990-91. Comunque una macchina AGA ha SICURAMENTE anche questo registro.

Avrete notato che dal workbench 2.0 in avanti è possibile scegliere il tipo di monitor, e settare una frequenza video anche "VGA" non interlacciata, ossia 640x480 a 31KHz, o anche 800x600 e altre risoluzioni particolari.

NTSC (525 linee, 227.5 colorclocks per scan line) 15Khz

PAL (625 linee, 227.5 colorclocks per scan line) 15Khz

VGA (525 linee, 114.0 colorclocks per scan line) 31Khz

Comunque per visualizzare queste risoluzioni serve un monitor almeno "VGA", oppure multisync/multiscan. La televisione e i monitor "normali" come il 1084 non riescono ad agganciare quelle frequenze.

Allora si potrebbe pensare: mi compro un monitor VGA/multisync almeno posso vedere sia la risoluzione PAL/NTSC che quella non interlacciata 31Khz! Purtroppo la maggior parte dei monitor che riesce a visualizzare il 640x480 a 31Khz non visualizza la risoluzione "televisiva" a 50/60Hz, per cui dovreste avere due monitor, uno per vedere una risoluzione e uno per vedere l'altra. Per questo occorre stare attenti! Se voleste comprare un monitor multisync/multiscan, accertatevi prima che visualizzi correttamente anche il 320x256 PAL dei videogiochi/demo, come fa il C= 1950, ad esempio.

Programmare i vari modi video 800x600 o simili è complicato, e non compatibile con tutti i monitor, per questo vedremo solo come fare il 640x480, supportato comunque anche dai peggiori monitor VGA del PC MSDOS.

Intanto vediamo un po di nuovi registri per la sincronizzazione:

```
VSSTRT          - Posizione linea verticale per VSYNC start.
VSSTOP          - Posizione linea verticale per VSYNC stop.
HSSTRT          - Posizione linea orizzontale per HSYNC start.
HSSTOP          - Posizione linea orizzontale per HSYNC stop.
HCENTER         - Posizione orizzontale per VSYNC nell'interlace.
```

E altri per il blanking programmabile:

```
HBSTRT          - Posizione linea orizzontale per HBLANK start.
HBSTOP          - Posizione linea orizzontale per HBLANK stop.
VBSTRT          - Posizione linea verticale per VBLANK start.
VBSTOP          - Posizione linea verticale per VBLANK stop.
```

I dati che abbiamo del nostro modo video sono:

```
VGA  (525 linee, 114.0 colorclocks per scan line) 31Khz
```

Dobbiamo quindi mettere in VTOTAL il numero di linee-1 (524) e in HTOTAL il numero di colorclocks per scanline-1 (113), più altri settaggi.

Per cambiare la frequenza orizzontale dai 15Khz (TV, monitor 1084), ai 31Khz dei monitor VGA/Multiscan/Multisync è necessario agire su un bel pò di registri, non solo il BEAMCON0 (che serve ad abilitare altri registri):

```
        LEA     $DFF000,A5

                ;5432109876543210
        MOVE.W  #%0001101110001000,$1DC(A5) ; BEACON0 - lista dei bit settati:

                        ; 3 - BLANKEN - COMPOSITE BLANK OUT TO CSY PIN
                        ; 7 - VARBEAMEN - VARIABLE BEAM COUNTER COMP. ENABLED
                        ;     Abilita i comparatori variabili di beam per
                        ;     operare nel contatore orizzontare principale,
                        ;     e disabilita lo stop hardware del display in
                        ;     orizzontale e in verticale.
                        ; 8 - VARHSYEN - VARIABLE HORIZONTAL SYNC ENABLED
                        ;     Attiva i registri HSSTRT/HSSTOP (var. HSY)
                        ; 9 - VARVSYEN - VARIABLE VERTICAL SYNC ENABLED
                        ;     Attiva i registri VSSTRT/VSSTOP (var. VSY)
                        ; 11- LOLDIS - DISABLE LONGLINE/SHORTLINE TOGGLE
                        ;     Disabilita lo scambio tra linee lunghe/corte.
                        ; 12- VARVBEN - VARIABLE VERTICAL BLANK ENABLED
                        ;     Attiva i registri VBSTRT/VBSTOP, e disabilita la
                        ;     "fine" hardware della finestra video.

        MOVE.W  #113,$1C0(a5)   ; HTOTAL - HIGHEST NUMBER COUNT, HORIZ LINE
                                ; Color clock massimo per linea orizzontale:
                                ; Il VGA ha 114 colorclocks per scan line!
                                ; Il valore va da 0 a 255: 113 va bene!

        MOVE.W  #%1000,$1C4(a5) ; HBSTRT - HORIZONTAL LINE POS FOR HBLANK START
                                ; I bit 0-7 contengono le posizioni di start
                                ; e di stop del blanking orizzontale in
                                ; incrementi di 280ns. I bit 8-10 servono per
                                ; un posizionamento a 35ns (1/4 di pixel).
                                ; In questo caso abbiamo settato 2240ns.

        MOVE.W  #14,$1DE(a5)    ; HORIZONTAL SYNC START - Numero di color
                                ; clocks per il Sync-start.

        MOVE.W  #28,$1C2(a5)    ; HORIZONTAL LINE POSITION FOR HSYNC STOP
                                ; Num. di color-clocks per Sync-stop.

        MOVE.W  #30,$1C6(a5)    ; HORIZONTAL LINE POSITION FOR HBLANK STOP
                                ; Linea orizzontale di stop Horiz BLANK

        MOVE.W  #70,$1E2(a5)    ; HCENTER - POS. ORIZZ. di VSYNCH in interlace
                                ; nel caso di beam counters variabili.

        MOVE.W  #524,$1C8(a5)   ; VTOTAL - HIGHEST NUMBERED VERTICAL LINE
                                ; Massima linea verticale numerata, ossia
                                ; la linea alla quale resettare il contatore
                                ; diposizione verticale.
                                ; Sappiamo che il modo VGA ha 525 linee.

        MOVE.W  #0,$1CC(a5)     ; VBSTRT - VERTICAL LINE FOR VBLANK START
        MOVE.W  #3,$1E0(a5)     ; VERTICAL SYNC START

        MOVE.W  #5,$1CA(a5)     ; VERTICAL LINE POSITION FOR VSYNC STOP
        MOVE.W  #29,$1CE(a5)    ; VBSTOP - VERTICAL LINE FOR VBLANK STOP

        MOVE.W  #%0000110000100001,$106(a5)     ; 0 - external blank enable
                                                ; 5 - BORDER BLANK
                                                ; 10-11 AGA dual playfiled fix
```

Ora basta puntare la nostra copperlist in *$dff080*, ricordandosi che il bit 0 del `BPLCON0` (*$dff100*) deve essere settato, e che se si vogliono più di 1 bitplane occorre abilitare il burst a 32/64 bit con `FMODE` (*$dff1fc*).

Ad esempio:

```
COPPERLIST:
        dc.w    $8E,$1c45       ; diwstrt
        dc.w    $90,$ffe5       ; diwstop
        dc.w    $92,$0018       ; ddfstrt
        dc.w    $94,$0068       ; ddfstop
        dc.w    $1e4,$100
```

```
7         dc.w    $108,0          ; modulo (non -8??)
8         dc.w    $10A,0
9
10                ; Puntate una figura 640x480.
11
12  BPLPOINTERS:
13        dc.w $e0,0,$e2,0        ; primo         bitplane
14        dc.w $e4,0,$e6,0        ; secondo          "
15        dc.w $e8,0,$ea,0        ; terzo            "
16        dc.w $ec,0,$ee,0        ; quarto           "
17        dc.w $f0,0,$f2,0        ; quinto           "
18        dc.w $f4,0,$f6,0        ; sesto            "
19        dc.w $f8,0,$fA,0        ; settimo          "
20        dc.w $fC,0,$fE,0        ; ottavo           "
21
22        dc.w    $100,$1241      ; bplcon0 (non settare bit hires, solo il
23                                  ; numero dei planes e i bit 0-9 e SHRES (6))
24
25  ; qua la palette
26
27        dc.w    $180,$000
28
29        dc.w    $1fc,$8003      ; sprite scan doubling???
30        dc.w    $FFFF,$FFFE     ; Fine Coplist
```

Vediamo un esempio pratico in `Lezione15i.s` (Se non avete un monitor capace di visualizzare 31Khz vedrete solo delle "strisciate").

Una nota: nessuno ha mai fatto una demo o un gioco a 31Khz, perché sono pochi gli utenti Amiga con un monitor VGA+. Se decideste di aggiungere l'opzione di visualizzare grafica in questa modalità, dovreste però prima far apparire una finestrella chiedendo se usare la frequenza normale o 31Khz!

15.14 Collisioni

Essendo stati aggiunti i bitplanes 7 e 8, occorreva un `CLXCON2` che potesse registrare le collisioni con questi 2 plane.

```
CLXCON2  $dff10e        - Extended collision control - controlla (se i
                          bitplane 7 e 8 sono inclusi nel detect!)
                          Questo registro è resetttato quando si scrive
                          nel vecchio CLXCON - La funzione dei bit è
                          analoga a quelli dei CLXCON

        BIT     NOME    DESCRIZIONE

        15-08           Non usati
        07      ENBP8   Abilita controllo bit plane 8
        06      ENBP7   Abilita controllo bit plane 8
        05-02           Non usati
        01      MVBP8   Match value per la collisione bitplane 8
        00      MVBP7   Match value per la collisione bitplane 8
Nota: disabilitare i bitplanes non previene le collisioni: se tutti i plane
      sono disabilitati, le collisioni sono "continue".
```

15.15 Blitter ECS+

Il blitter ha avuto dei potenziamenti già con l'ECS, ma per compatibilità è bene blittare sempre in modo OCS. Invece se si detecta l'AGA si è sempre sicuri di poter blittare ECS+, sempre che ci sia utile.

In pratica sono stati aggiunti il `BLTSIZV` (*$dff05c*) e il `BLTSIZH` (*$dff05E*), che sono, in pratica, due registri in cui mettere la grandezza VERTICALE e ORIZZONTALE della blittata, anziché

nel classico BLTSIZE ($dff058). Prima si deve scrivere nel BLTSIZV, poi nel BLTSIZH, e parte la blittata. Nel BLTSIZV va immessa l'altezza in linee, che può andare da 0 a 32767. Se si fanno delle blittate di seguito con la stessa altezza non occorre riscrivere nel BLTSIZV (*$dff05c*), rimane l'ultimo valore immesso. La blittata parte quando si scrive nel BLTSIZH (*$dff05e*), in cui occorre scrivere la grandezza orizzontale della blittata in word (da 0 a 2047, ossia fino a 32768 pixel). Mettere zero in questi 2 registri equivale al massimo, come il "vecchio" BLTSIZE. La massima blittata, dunque, è stata portata a 32768*32768, rispetto alla vecchia massima blittata di 1024x1024.

Ci sono poi un paio di cosucce meno importanti:

1. Il byte *$dff05b* (BLTCON0L) è un "fac simile" del byte LF dei minterms, ossia il byte basso di BLTCON0 (*$dff040*). Pare che renda alcune blittate leggermente più veloci, specialmente se il byte alto di BLTCON0 è sempre uguale e se ne cambia quello basso scrivendo qua... Comunque non ho notato velocizzazioni particolari.

2. Il bit 7 del BPLCON1 (*$dff042*), detto DOFF, quando è settato disabilita l'output dell blitter nel canale D. Questo comunque permette un input ai canali A, B e C o delle eventuali modifiche di indirizzo, senza che questo venga "scritto" nel canale D.

Spero di essere stato abbastanza chiaro e di aver detto tutto quello che serviva per programmare l'AGA. Ora non avete scuse! **dovete** fare qualcosa per il chipset AGA.

Comunque, se avete l'AGA avete anche un 68020+, quindi potrebbe risultarvi utile leggere la prossima lezione, che si occupa proprio di questo!

Parte II

Pratica

CAPITOLO 16

LICENZA D'USO

16.1 Introduzione

Tutti i sorgenti e gli esempi pubblicati possono essere utilizzati liberamente nei propri progetti, esattamente come era possibile farlo all'epoca della prima scrittura di questo corso di programmazione. Al giorno d'oggi esistono una moltitudine di possibilità per pubblicare il proprio codice tutelando al contempo il proprio lavoro dal "lamer" di turno.

Per la riedizione del 2016 di questo corso si è deciso "proteggere" le innumerevoli ore spese dalle molte persone che hanno contribuito alla sua realizzazione adottando per tutto il codice scritto una licenza di tipo *open source*. Questo non vuol dire che si sia deciso di limitarne in alcun modo il suo uso all'interno di future produzioni, significa solo che se tale codice troverà una sua utilità all'interno di demo, giochi o altro sarà necessario citare la fonte.

La licenza scelta è la *BSD 3-clause*, che non pone vincoli di riutilizzo anche all'interno di produzioni commerciali o l'inclusione in software di tipo closed source. L'unica "imposizione" è che venga sempre citata da qualche parte (che sia nei sorgenti, nella documentazione, in una scritta all'avvio del software ecc...) la provenienza del codice utilizzato.

Nella sezione seguente verrà riportata la licenza nella sua interezza ma non verrà ripetuta in tutti i pezzi di codice solo per una questione di editing. Si deve comunque trattare tutto il codice che non riporti altri tipi di licenza, come se fosse preceduto dal testo sotto riportato.

16.2 BSD 3-clause

CAPITOLO 17

LEZIONE 1

17.1 Lezione1a

```
; by Fabio Ciucci - Assemblare con "A", eseguire con "J"

Waitmouse:                          ; questa LABEL serve di riferimento per il bne.
        move.w        $dff006,$dff180        ; metti il valore di $dff006 nel $dff180
                                  ; cioe' di VHPOSR in COLOR00
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        Waitmouse        ; se no ritorna a waitmouse e ripeti
        rts                          ; esci

        END

; NOTA: il comando MOVE significa MUOVI, o meglio COPIA il numero contenuto
; nel primo operando nel secondo, in questo caso "LEGGI CHE NUMERO C'E' IN
; $DFF006, E METTILO IN $DFF180". Il .w significa che muove una word, cioe'
; 2 bytes, cioe' 16 bit (1 byte=8 bit, 1 word=16 bit, 1 longword=32 bit)
; NOTA2: il BTST seguito dal BNE serve per fare un salto nel programma nel
; caso che si sia verificata una condizione: si puo' tradurre cosi':
; BTST = CONTROLLA SE PIERO HA MANGIATO LA MELA,E SCRIVILO SU UN PEZZO DI CARTA
; BNE = IL BNE VA A LEGGERE NEL PEZZO DI CARTA SE PIERO HA MANGIATO LA MELA,
; COSA CHE NON PUO' VERIFICARE LUI, MA CHE GLI HA VERIFICATO L'AMICO BTST...
; NEL CASO CHE NEL FOGLIETTO CI SIA SCRITTO CHE NON LA HA MANGIATA, ALLORA
; SALTA ALLA LABEL INDICATA (IN QUESTO CASO BNE.S Waitmouse, quindi se piero
; non ha mangiato la mela il processore saltera' a Waitmouse e ripetera' il
; tutto; se invece la ha mangiata, allora non salta a Waitmouse, ma continua
; eseguendo l'instruzione sotto il BNE... in queso caso ci trova un RTS e
; di conseguenza il programma finisce. Il foglio in cui BTST scrive la sentenza
; per il BNE e' lo STATUS REGISTER, o SR. Se ad esempio al posto del BNE
; ci fosse stato un BEQ, allora il loop avverrebbe solo quando il mouse
; e' premuto, e finirebbe al suo rilascio (IL CONTRARIO: INFATTI BNE significa:
; BRANCH IF NOT EQUAL, ossia SALTA SE NON e' UGUALE (falso), mentre BEQ:
; BRANCH IF EQUAL, ossia salta se e' UGUALE (vero).
; Nella prima linea si legge il valore presente nel $dff006, ossia la linea
; raggiunta dal pennello elettronico che riscrive continuamente lo schermo,
```

```
; quindi un numero sempre diverso, e si mette nel $dff180, che e' il
; registro che controlla il colore 0, di conseguenza si ottiene lo schermo
; lampeggiante o striato, in cui cioe' viene cambiato il colore continuamente.
; verificate che $dff006 e' VHPOSR facendo "=c 006", e che $dff180 e' il
; COLOR 0, facendo "=C 180". questo aiuto lo potete chiedere all'ASMONE per
; ogni registro $dffxxx.
; il formato dei colori e' il seguente: $0RGB, cioe' la word del registro
; e' divisa in RED, GREEN e BLU, in 16 toni per colore; mischiandoli come
; dalla PALETTE o TAVOLOZZA del deluxe paint si puo' selezionare uno dei
; 4096 colori possibili (16*16*16=4096), ogni valore di RED, GREEN e BLU,
; ossia ROSSO, VERDE e BLU, va da 0 a F (numero esadecimale, ossia puo'
; essere 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f), per esempio provate a cambiare la
; prima linea con MOVE.W #$000,$dff180: si otterra' colore nero
; cambiandola con MOVE.W #$00e,$dff180 si otterra' il BLU,
; con un MOVE.W #$cd0,$dff180 si otterra' il GIALLO, ossia rosso+verde...
; provate a cambiare il colore per verificare se avete capito
; #$444 = grigio, #$900 = rosso scuro, #$e00 = rosso acceso, #$0a0 = verde....
; se infine cambiate il $dff180 con $dff182 lampeggeranno le scritte anziche'
; lo sfondo, cioe' quello colororato col colore 1. Se mettete entrambe le
; istruzioni una dopo l'altra lampeggeranno sia lo sfondo che le scritte.
; Il comando BTST controlla se un BIT in un dato indirizzo e'=0...
; ricordati che il numero dei bit va letto da destra verso sinistra e
; partendo da 0, ad esempio in un byte tipo %01000000 , il bit ad 1 e' il 6:
; 76543210                  5432109876543210
; 01000000    una word:  0001000000000000 <= qua e' a 1 il bit 12!!!
;
; P.S:        Il primo bit viene detto bit 0 e non bit 1, quindi non bisogna mai
;       confondersi per questa cosa, cioe' che, per esempio, il settimo bit
;       venga chiamato bit 6. Per non sbagliarvi mettete sempre la numerazione
;       ; 5432109876543210 sopra il numero in binario.
;
; il bit 6 di $bfe001 infatti e' il bottone sinistro del mouse.
; Il nome del registro $bfe001 e' CIAAPRA, ma nessuno se lo ricorda.
; il bottone destro invece e' il bit 2 di $dff016. prova a sostituire la
; linea BTST #6,$bfe001 con BTST #2,$dff016, e servira' il bottone destro
; per uscire dal ciclo. Fate tutte le variazioni suggerite per verificare!
; NOTA: se volete salvare il programma in modo che sia seguibile dal CLI
; basta fare "WO" dopo aver assemblato con A (e prima di fare il J!),
; e vi apparira' la finestrella per decidere dove salvarlo
; (Mi raccomando! salvatelo su un altro dischetto! Tenete il disco del
; corso protetto da scrittura e guai se ci scrivete sopra!!!).
; Se volete invece salvare il listato, usate il comando "W". ( su un altro
; disco!!!).

; PSPS: Avrete notato il BNE.S, che a un suffisso che non e' ne' .B, ne' .W
;       ne' .L!!!! Ebbene nelle istruzioni come BNE, BEQ, BSR si possono dare
;       solo due dimensioni: .B e .W, che pero' non influenzano il risultato,
;       infatti un bne.w fara' la stessa cosa di un bne.b. In queste istruzioni
;       e' permesso di chiamare il .B come .S, che sta per SHORT (corto), e
;       si puo' usare solo se la label a cui si riferisce non e' troppo
;       "lontana", altrimenti l'ASMONE durante l'assemblaggio la cambiera' nel
;       listato automaticamente in .W. Dato che il .S (che ripeto sta per .B)
;       puo' essere usato solo con tali istruzioni, credo sia meglio usarlo,
;       ora che lo sapete non dovrebbe creare problemi.
;       PSPSPS: Se mettete come grandezza un .L, (BNE.L) l'ASMONE non da errore
;       e assembla come .W, altri assemblatori danno errore. Se dimenticate
;       di mettere il suffisso (BNE Inizio), l'ASMONE assemblera' sempre come
;       BNE.W, lo stesso vale per le altre istruzioni! scrivere MOVE $10,$20,
;       non da errore perche' viene assemblato come MOVE.W $10,$20, MA NON E'
;       DETTO CHE TUTTI GLI ASSEMBLATORI SI COMPORTINO COSI', DUNQUE METTETE
;       SEMPRE IL SUFFISSO, che e' anche esteticamente piu' bello.
```

CAPITOLO 18

LEZIONE 2

18.1 Lezione2a

```
; Lezione2a.s - Questo programmino registra nel byte denominato col nome
; "contatore" il numero di volte che viene premuto il tasto destro, o meglio
; quanto e' stato premuto, infatti quando lo si tiene premuto viene continua-
; mente incrementato; per usecire si deve premere il tasto sinistro.

 Inizio:
        btst        #2,$dff016        ; POTINP - tasto destro del mouse premuto?
        beq.s        aggiungi        ; se si, vai ad "aggiungi"
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        inizio                ; se no, torna ad Inizio e ripeti tutto
        rts                        ; se si invece ESCI!

aggiungi:
        move.b        contatore,$dff180 ; metti il valore di CONTATORE in COLOR0
        addq.b        #1,contatore        ; Aggiungi 1 al valore di contatore
        bra.s        inizio                ; torna ad inizio e ripeti

contatore:
        dc.b        0                ; questo e' il byte che terra' il conto...

        END        ; Con END si determina la fine del listato, le parole
                Sotto l'END non sono considerate, e' come fossero tutte
                precedute dal ; (Punto e virgola)

NOTA: POTINP e' il nome del registro $dff016. Il nome in maiuscolo dopo
      il punto e virgola si diferisce sempre al nome del registro $dffxxx
In questo listato si nota l'uso delle label sia per rappresentare delle
istruzioni, (bne.s inizio, bra.s inizio), sia per rappresentare un byte
di dati (addq.b #1,contatore). Non c'e' differenza tra la label Inizio:
e la label contatore:, sono entrambe label, ossia NOMI CHE IDENTIFICANO UN
DATO PUNTO DEL PROGRAMMA, SIA ESSO UN BYTE, UN MOVE O QUALSIASI ALTRA COSA,
CHE SERVE PER FAR ESEGUIRE LE ISTRUZIONI CHE HA SOTTO, QUANDO PRECEDE DELLE
ISTRUZIONI (SI FARA' UN BNE LABEL, AD ESEMPIO), O A LEGGERE/SCRIVERE IL
```

BYTE, LA WORD O LA LONGWORD CHE PRECEDE. Ho notato che molti trovano
difficolta' ad entrare in questa logica. Facciamo degli esempi per chiarire
il ruolo delle LABEL: immaginate di avere un orticello rettangolare
recintato, con un viottolo che lo attraversa nel mezzo. Dopo averlo
vangato decidete di seminarci delle fragole, dell'insalata, del basilico e
del prezzemolo, quindi lo dividete in 4 rettangoli di diversa grandezza e
seminate. Per sapere da dove inizieranno a crescere i diversi ortaggi si
usano spesso quelle etichette di plastica con una punta da inserire nel
terreno, le avete presenti??? Allora piantiamole: l'orto ora ha 4 etichette
che spuntano dal terreno, ognuna con scritto sopra il nome dell'ortaggio: su
una troveremo FRAGOLE:, in un'altro INSALATA:, poi BASILICO: e PREZZEMOLO:.
Da notare che abbiamo messo le etichette nel punto dove inizia un tipo
di ortaggio, e di conseguenza dove finisce quello precedente:

```
FRAGOLE:        INSALATA:        BASILICO:        PREZZEMOLO:
  \/                \/                \/                \/
  ................oooooooooooooooooo^^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-_-_-_-_-_-_-_
```

Se considerate le fragole come i ".....", l'insalata come i "ooooo", il
basilico come i "^^^^^^^" e il prezzemolo come i "-_-_-_-_", noterete
che quando scrivo "BNE INSALATA" intendo VAI ALL'ETICHETTA INSALATA, non
"buttati nel mezzo dell'insalata" o "vai in direzione dell'insalata", ma
proprio "VAI AL CARTELLINO INFILATO NEL TERRENO IN CUI C'E' SCRITTO INSALATA,
ED ESEGUI LE ISTRUZIONI DOPO ESSO, in questo caso si eseguirebbero i "oooo".
Nel caso che usi la label in questo modo:

```
        addq.b        #1,BASILICO
```

Non faccio altro che aggiungere 1 seme nel primo byte dopo L'ETICHETTA, che
non ha cambiato funzione! Non indica contenuti o altre stranezze!!! Indica
sempre un punto della memoria, ossia del listato, che e' l'inizio del basilico.
Proviamo a fare un MOVE.B FRAGOLE,BASILICO

```
FRAGOLE:        INSALATA:        BASILICO:        PREZZEMOLO:
  \/                \/                \/                \/
  ................ooooooooooooooooo.^^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-_-_-_-_-_-_-_

  |                                  |
   \--> ----> ----> ----> ----> --->
```

Come noterete un ".", ossia il byte dopo fragole e' stato copiato nel byte
dopo BASILICO:, proviamo ora un MOVE.W INSALATA,FRAGOLE

```
FRAGOLE:        INSALATA:        BASILICO:        PREZZEMOLO:
  \/                \/                \/                \/
  oo..............ooooooooooooooooo.^^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-_-_-_-_-_-_-_

  ||              ||
   \<--- <--- <---/
```

Abbiamo spostato i primi 2 "oo" che erano dopo INSATATA: nei primi 2 byte
dopo FRAGOLE:

Se si volesse leggere o scrivere da un punto intermedio tra le LABEL, bastera'
aggiungerne un'altra dove si desidera: per mettere 4 bytes di INSALATA
nel centro del basilico dovremo fare una nuova LABEL denominata BAS2: nel
punto prestabilito, dopodiche' faremo un MOVE.L INSALATA,BAS2

Prima:

```
FRAGOLE:        INSALATA:        BASILICO: BAS2:        PREZZEMOLO:
  \/               \/                 \/          \/            \/
  oo.............ooooooooooooooooo.^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-_-_-_-_-_-_
```

Dopo:

```
FRAGOLE:        INSALATA:        BASILICO: BAS2:        PREZZEMOLO:
  \/               \/                 \/          \/            \/
  oo.............ooooooooooooooooo.^^^^^^^oooo^^^^^^_-_-_-_-_-_-_-_-_-_-_-_-_

                 ||||                          ||||
                  \ ----> ----> ----> -----> /
```

Abbiamo spostato i primi 4 bytes che si trovavano dopo INSALATA: nei primi 4 bytes dopo BAS2:

Il funzionamento e' lo stesso dello spostamento che avviene usando indirizzi reali, come gia' spiegato nella LEZIONE1 (CANE GATTO>NEO CANEO), soltanto che invece di operare con gli indirizzi, in cui ogni seme ha un indirizzo, si mettono delle etichette solo agli indirizzi che interessano:

Usando gli indirizzi:

```
FRAGOLE:        INSALATA:        BASILICO:        PREZZEMOLO:
  \/               \/                 \/                  \/
  ................ooooooooooooooooo^^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-_-_-_-_-_
  123456789012345678901234567890123456789012345678901234567890123456789012345
                   1111111111222222222233333333334444444444555555555566666666
```

Muovendosi con gli indirizzi, si possono copiare 4 byte di insalata da ogni posizione, ad esempio dalla n.15, e metterlo nella n.55: MOVE.L 15,55

```
FRAGOLE:        INSALATA:        BASILICO:        PREZZEMOLO:
  \/               \/                 \/                  \/
  ................ooooooooooooooooo^^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-oooo_-_-_-_
  123456789012345678901234567890123456789012345678901234567890123456789012345
                   1111111111222222222233333333334444444444555555555566666666
                         \--> ---> ---> ---> ---> ---> ---> --->/
```

La stessa operazione si puo' pero' fare mettendo una label alla posizione 15 e un'altra alla posizione 55, facendo poi "move.l LABEL1, LABEL2"

```
                      LABEL1:                                      LABEL2:
                       \/                                        \/
  ................ooooooooooooooooo^^^^^^^^^^^^^^^^^^_-_-_-_-_-_-_-oooo_-_-_-_
                        \---> ---> ---> ---> ---> ---> ---> --->/
```

Come mai si e' preferito usare le LABEL piuttosto che gli indirizzi? SEMPLICE! Perche' usando gli indirizzi, se avessimo inserito delle cose tra l'insalata e il basilico, la destinazione non sarebbe stata piu' 55, ma un altro numero, per esempio 80, e avremmo dovuto cambiare tutti i numeri spostandoli in avanti per farci entrare il nuovo pezzo inserito. Invece, con le label, se inseriamo qualcosa tra l'una e l'altra non ci comporta modifiche, perche' l'ASMONE calcola ogni volta l'indirizzo della label.

Provate ad eseguire questo prog, la prima volta senza premere il tasto destro, ma solo il tasto sinistro per uscire: il byte CONTATORE in questo caso e' rimasto 0, come si puo' verificare con il comando M, che visualizza i valori

effettivi contenuti negli indirizzi di memoria in formato di bytes in
esadecimale (indicato con il numero della locazione, ad esempio M $50000, o
il nome di una label): facendo M contatore si otterra' uno ZERO, seguito
da altri numeri che corrisponderanno ai byte seguenti in memoria che non ci
interessano. (per avanzare in memoria premete varie volte return, per terminare
premete il tasto ESC - i byte sono ovviamente in formato ESADECIMALE)
Riassemblate con A ed eseguitelo una seconda volta, questa
volta premendo il tasto destro prima di uscire (col sinistro): rifacendo
"M contatore" si otterra' un numero diverso da zero, che corrispondera'
al numero di cicli in cui il tasto destro del mouse era premuto, infatti
il ciclo viene eseguito molto velocemente dal processore e anche premendo il
tasto destro per un istante si ottengono numeri superiori a 1.
Va notato che il contatore in questione e' lungo un byte, quindi puo'
raggiungere un valore massimo di 255, ossia $FF, ossia %11111111 in binario,
cioe' tutti gli otto bit che formano il byte ACCESI (1), dopodiche' il numero
ripartira' da zero (se si continua ad addizionare). ($ff+1 = 0, $ff+2 = 1...)
Il passo avanti di questo programmino rispetto al primo e' che la struttura
dei salti condizionati e' piu' complessa (E mi raccomando di non andare avanti
fino a che non la avete capita!), inoltre viene usato un byte come variabile.
Questo byte nominato CONTATORE non solo viene scritto, ma viene anche letto
per scriverne il valore nel $dff180, ossia il COLOR0: di qui si intuisce come
possano essere gestite molte VARIABILI, ossia bytes, words o longwords in
cui vengono scritti e letti numeri utili al programma, ad esempio il numero
delle vite del PLAYER 1, la sua energia, i suoi punti, eccetera.
L'uso delle LABEL e' utile al programmatore ma il programma una volta
assemblato, diviene una serie di bytes, che se sono letti dal 68000
vengono interpretati come istruzioni che si riferiscono ad indirizzi diretti:
per verificare cio, assemblate il programma e fate un D Inizio...
Verra' cosi' visualizzato il programma assemblato in memoria nella sua vera
forma, in cui al posto delle label compaiono gli indirizzi EFFETTIVI: come
osservate la prima colonna di numeri a sinistra sono gli indirizzi della
memoria dove stiamo leggendo, la seconda colonna di numeri sono i comandi
nella loro forma REALE in memoria, cioe' sequenze di bytes (ad esempio la
prima riga BTST #2,$dff016 in memoria diventa 0839000200dff016, in cui
$0839 significa BTST, 0002 e' il #2, 00dff016 e' l'indirizzo interessato)...
la terza colonna, quella a destra, riporta il DISASSEMBLATO, ossia fa il
contrario di quando assembla: trasforma i BYTES in ISTRUZIONI (quando si
preme A (assembla) invece vengono trasformate istruzioni dal formato MOVE,ADD,
BNE,BTST... in BYTES). Leggendo noterete subito che le label sono sostituite
dagli indirizzi reali dove si trovano routines o variabili.
Come ulteriore verifica del fatto che le istruzioni diventano numeri ben
precisi, sostituite la linea:

```
        btst        #2,$dff016        ; POTINP - tasto destro del mouse premuto?
```

Con la linea equivalente:

```
        dc.l        $08390002,$00dff016
```

oppure:

```
        dc.w        $0839,$0002,$00df,$f016
```

oppure:

```
        dc.b        $08,$39,$00,$02,$00,$df,$f0,$16
```

In tutti i casi il risultato e' un 0839000200dff016 in memoria, che il 68000
interpreta come "btst #2,$dff016", cioe' "il bit 2 di $dff016 e' a zero?".

Se la variabile fosse stata una WORD anziche' un BYTE, il listato dovrebbe
essere modificato cosi':

```
Inizio:
        btst        #2,$dff016        ; POTINP - tasto destro del mouse premuto?
        beq.s        aggiungi        ; se si, vai ad "aggiungi"
        btst        #6,$bfe001        ; stato sinistro del mouse premuto?
        bne.s        inizio                ; se no, torna ad Inizio e ripeti tutto
        rts                        ; se si invece ESCI!

aggiungi:
        move.w        contatore,$dff180 ; COLOR0 - Usare .w invece che .b
        addq.W        #1,contatore        ; usare ADDQ.W invece che ADDQ.B!!!
        bra.s        inizio

contatore:
        dc.W        0                ; dc.w invece di dc.b (lo stesso che dc.b 0,0)
```

In questo caso il numero massimo contenibile da una word prima che ricominci
da capo e' $FFFF, ossia 65535, ossia %1111111111111111.

Se si volesse usare una LONGWORD per CONTATORE:, il numero massimo prima
di riazzerarsi sarebbe $FFFFFFFF , ossia qualche miliardo, ma bisogna
considerare che il bit alto (ossia il trentunesimo nel caso della longword)
e' usato per il segno del numero: provate a fare ?$0FFFFFFF e otterrete
un 268 milioni e rotte, ed in binario si nota che i quattro bit piu' alti
del numero (ossia i primi quattro dopo il %) sono a zero. il massimo numero
positivo che si puo' ottenere e' $7FFFFFFF ossia in binario:

```
        ;10987654321098765432109876543210        ; numero di bit da 0 a 31
        %01111111111111111111111111111111
```

Infatti il bit 31 (che sarebbe il trentaduesimo, ma e' il trentuno perche'
conta anche lo zero) e' a ZERO, mentre gli altri sono tutti ad 1.
Se si fa ?$7FFFFFFF+1 si ottiene -2 miliardi e rotte, e mano mano che si
aumenta il numero si avvicina allo zero (-1 miliardo, -100 milioni, -10 etc)
infatti se si fa un ?-1 si ottiene $FFFFFFFF, con ?-2 invece $FFFFFFFE.

Questo sistema del bit alto usato come segno puo' essere valido anche per
i byte e le word: per i byte, un move.b #-1,$50000 si puo' scrivere anche
move.b $FF,$50000 quindi il massimo numero positivo diventerebbe:
 %01111111 , ossia $7f, 127. Per le word il massimo numero positivo diventa
 %0111111111111111, ossia $7FFF, ossia 32767. Comunque e seconda di come si
fa il programma i numeri possono essere usati come numeri positivi e negativi
o come numeri assoluti.

Provate a cambiare il listato in modo che CONTATORE: sia una word, come
descritto sopra: Potete usare le funzioni dell'editor di ASMONE, il
cosiddetto TAGLIA ed INCOLLA: per "ritagliare" un pezzo di testo e
copiarlo in un altro punto, premete insieme il tasto Amiga destro+b
all'inizio della parte di testo che volete copiare; in questo caso selezionate
il sorgente modificato a WORD sotto la linea "essere modificato cosi':"
posizionandovi appunto sopra la label Inizio: e premendo Amiga+b. Ora
potete selezionare il blocco (che vi apparira' in negativo), spostandovi in
basso con il cursore. Arrivati sotto il Dc.W 0 premete Amiga+c, e il pezzo di
testo comprendente il listato andra' in memoria. Ora andate in cima al
listato a colpi di CURSORE SU+SHIFT, e premete Amiga+i ...
Magicamente apparira' una copia del testo che avevate selezionato prima.
A questo punto basta mettere un END (distanziato dall'inizio della riga con
degli spazi, o meglio con un TAB) sotto il DC.W 0 per escludere il primo
sorgente con CONTATORE: lungo un byte. Assemblate e Jumpate.

P.S: Non fate caso, per ora, a quella sfilza di numeri, a volte evidenziati,
     che compaiono dopo ogni "J", il loro significato verra' spiegato dopo.

Subito noterete la differenza di lampeggiamento dello schermo quando premete
```
```

il tasto destro; provate a fare un M CONTATORE adesso:
ora e' una WORD, quindi saranno valide le prime 2 coppie di numeri,
ossia i primi 2 bytes... se compariranno ad esempio 00 30 significa che
l'ADDQ.W #1,CONTATORE e' stato eseguito $30 volte, ossia 48 volte; un valore
di 02 5e significherebbe $25e volte, ossia 606 volte.

Se non siete esperti di editor di testi fate un po di prove di TAGLIA E INCOLLA
scopiazzando e inserendo parti di questo testo qua' e la', e considerate che
se una volta selezionato un blocco con Amiga+b invece di copiarlo in memoria
con Amiga+c, premete Amiga+x il blocco selezionato sara' anche cancellato, ma
premendo Amiga+i si potra' reinserire da un'altra parte. Vi assicuro che
programmare e' tutto un TAGLIA e INCOLLA, in quanto questo trucchetto permette
di risparmiare il tempo di riscrivere parti di programma simili, che invece
possono essere copiate e modificate velocemente.

Bisogna districarsi bene nella numerazione binaria per poter lavorare, infatti
anche molti registri hardware sono BITMAPPED, ossia ogni bit corrisponde ad
una funzione. Ecco una tabella per rendere piu' chiara la differenza:

ESADECIMALE	BINARIO	DECIMALE
0	%00000	0
1	%00001	1
2	%00010	2
3	%00011	3
4	%00100	4
5	%00101	5
6	%00110	6
7	%00111	7
8	%01000	8
9	%01001	9
$A	%01010	10
$B	%01011	11
$C	%01100	12
$D	%01101	13
$E	%01110	14
$F	%01111	15
$10	%10000	16
$11	%10001	17
$12	%10010	18
...	...	...

Come vedete il binario segue una logica semplice di riempimento con gli 1 fino
a che non si giunge a 11, 111, 1111, 11111, 111111, eccetera, cioe' fino a che
non scatta la cifra seguente: dopo %011 c'e' %0100, dopo %0111 c'e' %01000,
dopo %01111 c'e' %010000, dopo %011111 c'e' %0100000 e cosi' via.
Va ricordato che i numeri in formato esadecimale sono preceduti dal segno del
dollaro $, mentre quelli binari sono preceduti dal segno di percentuale %.
I numeri in formato normale decimale invece non devono essere preceduti da
nessun segno. Se per esempio scriviamo 9 o $9, intendiamo sempre 9, ma se
invece scriviamo 10 o $10 intendiamo 10 in decimale o 16 in esadecimale!
Ricordatevi dunque che dopo il 9 un $ in piu' o in meno cambia tutto.
Non e' importante saper convertire a memoria i numeri, quello non serve perche'
basta usare il comando "?" per eseguire qualsiasi operazione o conversione,
infatti il risultato lo da in tutti i formati, decimale, esadecimale, binario
e ASCII, ossia in forma di CARATTERI. Infatti i caratteri come "abcd" non sono
altro che bytes, che secondo lo standard ASCII simile nei vari computers
corrispondono a certi caratteri, ad esempio "a"= $61, mentre "A"= $41.
Potete verificarlo facendo un ?"a", oppure un ?$61 dalla linea comandi.

NOTA: Il .s al BNE significa SHORT, ossia "corto" (equivalente a .b)
anziche' .s, oppure .w quando la label a cui si riferisce e' piu' lontana.
Provate sempre a mettere il .s al BNE, e vedrete che se la label indicata

```
dopo il beq.s o il bne.s e' troppo distante (piu' di 127 bytes), viene corretto
dall'assembllatore in .w. Per verificarlo fate questa modifica:

Inizio:
        btst        #2,$dff016        ; POTINP - tasto destro del mouse premuto?
        beq.s        aggiungi        ; se si, vai ad "aggiungi"
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        inizio                ; se no, torna ad Inizio e ripeti tutto
        rts                        ; se si invece ESCI!

        dcb.b        200,0                ; questa direttiva sara' spiegata in
                                ; seguito, in questo caso mette 200
                                ; bytes $00 in memoria, aumentando la distanza
                                ; fra le label Inizio: e aggiungi:

aggiungi:
        move.b        contatore,$dff180 ; metti il valore di CONTATORE in COLOR0
        addq.b        #1,contatore        ; Aggiungi 1 al valore di contatore
        bra.s        inizio                ; torna ad inizio e ripeti

Assemblando, vedrete che l'assemblatore vi segnala dei FORCED TO WORD SIZE,
infatti ha FORZATO A WORD i bne.s, proprio nel listato, perche' la distanza
tra Inizio: e aggiungi: era maggiore di 128. Vi consiglio di mettere sempre
il .s dopo i bra, bsr, beq, bne e simili, l'assemblatore correggera' quando
serve. Si puo' anche mettere sempre .w, ma le istruzioni .s sono piu' veloci
e occupano meno bytes. Per riprendere il concetto di LABEL discusso prima,
questo del dcb.b 200,0 inserito e' un esempio lampante dell'utilita' delle
LABEL, che ci hanno evitato di riscrivere la nuova posizione assunta da
aggiungi:, ossia 200 bytes piu' avanti.
```

18.2 Lezione2b

```
; Lezione2b.s

Inizio:
        MOVE.L        CANE,GATTO
        rts

CANE:
        dc.l        $123456

GATTO:
        dc.l        0

        END

Con questo esempio si puo' verificare che una volta assemblato il sorgente
al posto delle label viene assemblato l'indirizzo effettivo di CANE e GATTO:
Assemblate con A, dopodiche' fate "D Inizio" e noterete la sostituzione con
gli indirizzi. Dopo l'rts noterete un paio ORI.B e altre istruzioni: in
realta' queste sono un tentativo di interpretazione delle 2 longword CANE e
GATTO, infatti dopo il $4e75 dell'rts, noterete il $00123456, ossia la prima
longword denominata da noi CANE, e $00000000, ossia GATTO.
Ora eseguitelo con J, dopodiche' fate un M GATTO, e vi apparira' 00 12 34 56
infatti la longword contenuta in CANE e' stata copiata in GATTO:
Ora modificate la prima linea in MOVE.L #CANE,GATTO, assemblate e fate
"D inizio" ... constaterete che anche questa volta sono state sostituite
le label col loro indirizzo reale, infatti l'unica differenza con la prima
prova e' l'aggiunta del cancelletto (#). Ma questo cambia tutto come dal giorno
alla notte!!!!!!! Infatti questa volta se eseguite col J e fate M GATTO
```

```
verificherete che c'e' l'indirizzo di CANE! ossia, se l'istruzione fosse stata
assemblata come MOVE.L #$34200,$34204 , dopo l'esecuzione sarebbe stato
inserito in $34204 (cioe' GATTO) il numero FISSO dopo il cancelletto (#),
ossia l'indirizzo di CANE, ossia $34200.
Riassunto finale:

        MOVE.b        $10,$200        ; copia il valore .b contenuto nell'indirizzo
                                  ; $10 nell'indirizzo $200

        MOVE.b        #$10,$200        ; mette il numero $10 nella locazione $200
        MOVE.B        #16,$200        ; come sopra, infatti $10 = 16!!!
        MOVE.B        #%10000,$200        ; come sopra, infatti %10000 = 16!!!

NOTA: ASMONE alloca, ossia posiziona il programma ogni volta in un indirizzo
diverso, lo stesso fa il sistema operativo quando caricate un programma, a
seconda di dove avete memoria libera. Questo sistema e' uno dei punti di
forza del sistema multitasking AMIGA. Quando salvate il file eseguibile con WO
salvate il file nel formato AMIGADOS, che poi il sistema operativo mettera'
in memoria dove meglio credera'. Per questo scrivo "se fosse a $34200....":
perche' puo' essere assemblato a qualsiasi indirizzo.
Provate a caricare programmi in multitasking prima dell'asmone e noterete meno
memoria allocabile alla selezione iniziale (ALLOCATE Chip, fast...) e che
facendo D Inizio la locazione e' piu' alta, infatti la memoria sottostante
e' gia occupata. Programmare ad indirizzi FISSI, ossia specificando sempre
l'indirizzo anziche' la label non va fatto per giochi o demo in cui si
puo' uscire ritornando al workbench, perche' se ad esempio si definisce
lo schermo del gioco a $70000 e a quella locazione e' gia presente un
programma caricato prima, all'uscita otterremo un bel GURU MEDITATION, detto
COMA... se non volete far andare l'Amiga in coma continuamente dunque
programmate come questo corso insegna. Indirizzi fissi si possono, o alle
volte si DEVONO usare se si fanno giochi o demo in AUTOBOOT, cioe' che
partono non dal WB ma automaticamente e la cui directory non puo' essere
vista (molti giochi sono cosi'). Prima di fare un gioco o una demo in
autoboot credo sia meglio imparare almeno a visualizzare qualcosa sullo
schermo pero', quindi ne parlero' piu' avanti.
```

18.3 Lezione2c

```
; Lezione2c.s

Inizio:
        LEA         CANE,a0
        MOVE.L        #CANE,a1
        MOVE.L        CANE,a2
        move.l        a0,GATTO1
        move.l        a1,GATTO2
        move.l        a2,GATTO3
        rts

CANE:
        dc.l        $12345678

GATTO1:
        dc.l        0

GATTO2:
        dc.l        0

GATTO3:
        dc.l        0
```

```
        END
```

Assemblate, fate un D Inizio per controllare a che indirizzo sono allocate
le label, dopodiche' eseguite con J. Verificherete gia' che dopo il J la
lista dei registri riporta dei numeri in negativo, e precisamente:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: NUMERO   NUMERO   12345678 00000000 00000000 00000000 00000000 NUMERO (SP)
SSP= ..... USP= SR.....
```

Ogni volta che si e' eseguito un listato vengono visualizzati tutti i
registri: la prima fila e' quella dei D0,D1,D2,D3,D4,D5,D6,D7, la seconda
fila e' quella a0,a1,a2,a3,a4,a5,a6,a7.
Sotto ci sono altri registri di cui parleremo piu' aventi. Il numero in A7
e' lo SP attuale, non interessa in questo momento. Controllate invece i
numeri presenti in A0,A1 ed A2: il primi sono 2 numeri uguali, in questo caso
l'indirizzo di CANE:, infatti le due istruzioni:

```
        LEA         CANE,a0                ; piu' veloce di MOVE.L #CANE,A1! fate cosi'!
        MOVE.L         #CANE,a1
```

Fanno la stessa cosa, cioe' copiare l'indirizzo della label in un registro.
In A3 invece si legge 12345678, ovvero il contenuto della longword CANE:
infatti l'istruzione MOVE.L CANE,a2 mette il contenuto della long CANE in a2.
Come ulteriori verifiche, controllate dopo il J con un M CANE, e constaterete
che CANE e' alla stessa locazione che appare in a0 e a1; dopodiche' potete
anche controllare con M GATTO1 e M GATTO2 che queste 2 longword contengono
l'indirizzo di gatto, infatti ci viene copiato con queste 2 istruzioni:

```
        move.l         a0,GATTO1
        move.l         a1,GATTO2
```

Infine con M GATTO3 si verifichera' che contiene il contenuto della long
GATTO, ossia $12345678.
Per fare questi 3 controlli in un colpo solo potete fate un m gatto1 e
premere il tasto return (o invio o "A CAPO!") varie volte: otterrete nei
primi 4 bytes l'indirizzo di CANE, nei 4 bytes seguenti lo stesso indirizzo,
nei 4 bytes seguenti il contenuto .L di cane, ossia $12345678. Continuando si
vedranno numeri che non c'entrano nulla, infatti state vedendo una parte
di memoria vuota od occupata da chissa' che cosa.
Se volete fare qualche modifica, potete aggiungere per esempio prima dell'RTS
queste linee:

```
        MOVE.L         A0,D0
        MOVE.L         A1,D1
        MOVE.L         A2,D2
```

Ed otterrete nella lista dei registri dopo il J un cambiamento anche dei
primi 3 registri DATI.
NOTA1: Come avete visto usare il LEA e' meglio di fare un MOVE.L #lab,a0, ma
fate attenzione! il comando lea puo' essere usato solo per mettere un
valore nei registri indirizzo! non si puo' fare ad esempio LEA LABEL,d0!!!!
per mettere l'indirizzo di una label in un registro dati o in un altra long
della memoria si deve usare il MOVE.L #LABEL,destinazione!!!!
NOTA2: Di solito nei registri a0,a1,a2... si mettono indirizzi e nei
registri d0,d1,d2,d3... si mettono dati vari, ma spesso si mettono indirizzi
nei registri dati (d0,d1...) o dati nei registri indirizzi (a0,a1,a2..), a
seconda della situazione. Insomma per darvi un idea del loro utilizzo,
vengono usati come un foglio degli appunti in cui tenete un certo numero
di numeri di telefono o dove riportate quanto avete speso per comprare il
gelato, quindi come utili E VELOCI longword a disposizione che possono

essere usate a volonta', basta ricordarsi cosa ci avete messo!!!!!

18.4 Lezione2d

```
; Lezione2d.s

Inizio:
        lea        CANGURO,a0        ; in A0 mettiamo l'indirizzo di CANGURO
        move.l        (a0),d0               ; in d0 mettiamo il valore .L che troviamo
                                ; all'indirizzo che e' in a0, ovvero
                                ; la prima longword contenuta in CANGURO
        move.l        CANGURO,d1        ; in d1 mettiamo il contenuto della prima
                                ; longword (4 bytes=4 indirizzi) di canguro
        move.l        a0,d2                ; in d2 mettiamo il numero contenuto in a0,
                                ; ossia l'indirizzo di CANGURO caricato
                                ; prima col LEA CANGURO,a0
        move.l        #CANGURO,d3        ; in d3 mettiamo l'indirizzo di CANGURO
        rts

CANGURO:
        dc.l        $123

        END

Con questo esempio si nota la differenza tra l'indirizzamento diretto, quello
indiretto e quello assoluto: una volta assemblato , fate un D Inizio per
controllare e dopo averlo eseguito col J si notera' il risultato nei
registri: in d0 ed in d1 noterete $123, ossia il contenuto .L di CANGURO:

        lea        CANGURO,a0        ; in A0 mettiamo l'indirizzo di CANGURO
        move.l        (a0),d0               ; in d0 mettiamo il valore contenuto
                                ; nell'indirizzo che e' in a0, ovvero
                                ; il valore .L contenuto in CANGURO
                                ; (Con il MOVE.L si copia il byte contenuto
                                ; nell'indirizzo in a0, nonche' i 3 seguenti,
                                ; essendo una long lunga 4 bytes)
E' equivalente a:

        move.l        CANGURO,d1        ; in d1 mettiamo il contenuto .L di canguro

Infatti in tutti e due i casi il contenuto .L di canguro va nel registro dati.

Invece in d2,d3 ed a0 si notera' l'indirizzo di CANGURO, infatti:

        lea        CANGURO,a0        ; in A0 mettiamo l'indirizzo di CANGURO
        move.l        a0,d2                ; in d2 mettiamo il numero contenuto in a0,
                                ; ossia l'indirizzo di CANGURO caricato col LEA

E' equivalente a:

        move.l        #CANGURO,d3        ; in d3 mettiamo l'indirizzo di CANGURO

Queste differenze di indirizzamento devono essere chiare, infatti una volta
che si conoscono basta ricordarsi i comandi, i quali usano tutti lo stesso
sistema di indirizzamento.

Esempi di indirizzamenti fin ora analizzati:

DIRETTO:
        move.l        a0,a1
```

```
INDIRETTO:
        clr.l        (a0)
        move.l        (a3),(a4)

ASSOLUTO:
        move.l        #LABEL,d0
        MOVE.L        #10,d4
```

18.5 Lezione2e

```
; Lezione2e.s

Inizio:
        lea        $dff006,a0        ; metti $dff006 (VHPOSR) in a0
        lea        $dff180,a1        ; metti $dff180 (COLOR0) in a1
        lea        $bfe001,a2        ; metti $bfe001 (CIAAPRA) in a2
Waitmouse:
        move.w        (a0),(a1)        ; metti il valore di $dff006 nel color 0
        btst        #6,(a2)               ; tasto sinistro del mouse premuto?
        bne.s        Waitmouse        ; se no ritorna a waitmouse e ripeti
        rts                         ; esci

        END
```

Come si puo' notare, il ciclo Waitmouse e' composto di indirizzamenti indiretti anziche' diretti, ovvero invece di operare direttamente con gli indirizzi, si mettono questi ultimi in registri dati e si leggono o si scrivono usando i registri tra parentesi (indirettamente).
Questo aumenta la velocita' del loop, in quanto sono piu' veloci i registri degli indirizzamenti diretti (si dovrebbe notare infatti un leggero cambiamento nel lampeggiamento dello schermo rispetto a LEZIONE1a.s dovuto alla maggiore velocita' di esecuzione).
Dopo l'esecuzione si puo' verificare che i registri a0,a1 ed a2 contengono rispettivamente $dff006 (VHPOSR), $dff180 (COLOR00) e $bfe001 (CIAAPRA).
Se si vogliono proprio togliere tutti i numeri dal loop, si puo' modificare il BTST #6,$bfe001 con un BTST d0,$bfe001, dove in d0 ci sia 6:

```
Inizio:
        lea        $dff006,a0        ; VHPOSR - metti $dff006 in a0
        lea        $dff180,a1        ; COLOR00 - metti $dff180 in a1
        lea        $bfe001,a2        ; CIAAPRA - metti $bfe001 in a2
        moveq        #6,d0                ; metti il valore 6 in d0
Waitmouse:
        move.w        (a0),(a1)        ; metti il valore di $dff006 nel color 0
        btst        d0,(a2)               ; tasto sinistro del mouse premuto?
                                ; (bit 6 del $bfe001 azzerato?)
        bne.s        Waitmouse        ; se no ritorna a waitmouse e ripeti
        rts                         ; esci
```

(usate Amiga+b,c ed i per sostituire questo sorgente a quello sopra)

NOTA: Per mettere un 6 in d0 anziche' MOVE.L #6,d0 ho usato MOVEQ #6,d0, perche' i numeri sotto il $7f (ossia 127), sia negativi che positivi, possono essere immessi nei registri dati con lo speciale comando MOVEQ, che e' sempre .L, quindi non si mette .b,.w o .l come gia' visto per il LEA.
Esempi:

```
        MOVEQ        #100,d0        ; meno di 127
        MOVE.L        #130,d0        ; piu' di 127, va usato il move.l normale.
        MOVEQ        #-3,d0        ; fino a -128 si puo' usare il MOVEQ.
```

```
NOTA2: e' molto comune usare tutti i registri mettendoci gli indirizzi ed i
dati necessari alle routines (sottoprogrammi), perche' sono piu' veloci
e flessibili. Ma risultano ovviamente meno leggibili, infatti immaginate
che gli indirizzi e i dati siano stati messi nei registri all'inizio
del programma e che ci troviamo in mezzo ad esso con questa routine:

Routin:
        move.w        (a0),(a1)
        btst         d0,(a2)
        bne.s         Routin

In cui vediamo solo degli (a0),(a1), dei d0,(a2) eccetera. SE NON SAPPIAMO
COSA C'E' NEI REGISTRI TUTTO QUESTO APPARIRA' INSENSATO, quindi vi assicuro
che e' importante impararsi gli indirizzamenti e ricordarsi in che punto
mettete gli indirizzi e i dati nei registri per poter capire quello che si
e' scritto anche solo na settimana prima; la forza dei processori 68000 sta
proprio nella capacita' di lavorare INDIRETTAMENTE alla memoria usando
i registri nei loro diversi indirizzamenti, ma in questo consiste anche
la difficolta'.

Per esercitarvi provate a scrivere dei listati inutili ingarbugliati
tentando di capire in fondo al listato cosa risulti, cosa che potrete poi
verificare eseguendolo. Vi suggerisco un inizio, continuate voi
l'ingarbugliamento come se risolveste un quesito enigmistico:

        lea         PUFFO,a0
        move.l        (a0),a1
        move.l        a1,a2
        move.l        (a2),d0
        moveq        #0,d1
        move.l        d1,a0
        .....
        rts

PUFFO:
        dc.l         $66551
```

18.6 Lezione2f

```
; Lezione2f.s

Inizio:
        lea         START,a0        ; metti in a0 l'indirizzo da dove iniziare
                                    ; ovvero in a0 va l'indirizzo di START, ossia
                                    ; DOVE si trova START, non cosa contiene!
        lea         THEEND,a1         ; metti in a1 l'indirizzo dove finire
                                    ; Ossia si mette l'indirizzo della fine dei
                                    ; 40 bytes, infatti sta SOTTO i 40 bytes.
                                    ; Ora, TUTTO QUELLO CHE SI TROVA TRA la label
                                    ; START: e la LABEL THEEND: verra' pulito
                                    ; da LOOP CLELOOP:, siano essi 40 bytes
                                    ; o di piu', anche se ci mettete delle
                                    ; istruzioni.

CLELOOP:
        clr.l         (a0)+         ; Azzera la long in (a0), poi aggiungi 4 ad a0 (long!)
                        ; ATTENZIONE! Questo e' un indirizzamento indiretto,
                        ; in cui non si cancella il registro a0, ma il
                        ; contenuto dell'indirizzo, ossia 4 $FE alla volta,
```

```
                        ; ($fe e' un numero a caso che ho messo tanto per
                        ; esempio per distinguerlo dagli zeri! per dimostrare
                        ; che pulisco parto da una zona riempita con degli $FE;
                        ; essendoci un + dopo la parantesi, ogni volta che
                        ; viene eseguita il valore di a0 aumenta di 4, ossia
                        ; si mette al prossimo indirizzo da pulire)
                        ; (Il primo passaggio vengono azzerati i primi 4
                        ; $FE sotto start, il secondo passaggio i 4 seguenti
                        ; e cosi' via). da notare che aumenta soltanto a0,
                        ; mentre a1 rimane fermo all'indirizzo THEEND.
        cmp.l       a0,a1       ; a0 e' uguale ad a1? Cioe' siamo all'indirizzo THEEND?
                        ; (Infatti a0 aumenta di 4 ogni ciclo, e fermiamo il
                        ; ciclo quando a0 raggiunge l'indirizzo THEEND)
        bne.s       CLELOOP       ; se no, torna ad eseguire CLELOOP...
        rts               ; ESCI dal prog e torna ad ASMONE

START:
        dcb.b       40,$fe        ; Il comando DCB serve per mettere in memoria un
                        ; numero definito di byte, word o long uguali tra
                        ; di loro: similmente al comando DC.B, in cui in
                        ; questo caso avremmo dovuto fare dc.b $fe,$fe,$fe...
                        ; mettendo 40 $fe. Invece col comando dcb possiamo
                        ; fare piu' semplicemente dcb.b 40,$fe, ossia
                        ; METTI QUA IN MEMORIA 40 bytes $fe.
THEEND:                 ; questa label segna la fine dei 40 bytes...

        dcb.b       10,0        ; mettiamo 10 bytes azzerati qua tanto per sfizio

        end
```

Attenzione! con LEA START,a0, in a0 c'e' l'indirizzo del primo dei 40 bytes $fe, e non contiene i 40 bytes!!! La LABEL e' una convenzione usata nella programmazione per orientarsi nel listato, infatti servono per dare un nome alle varie parti del programma, siano esse istruzioni o altro, poi riferendoci a quella LABEL ci riferiremo AL PUNTO ESATTO IN CUI E' MESSA LA LABEL, cioe' l'indirizzo dove sta la label; per evitare confusioni, immaginatevi come mai sono state inventate le LABEL: se non ci fossero state, avremmo dovuto numerare ogni byte, ossia ragionare per indirizzi, ad esempio unvece di fare un BNE.S CLELOOP avremmo dovuto scrivere direttamente BNE.S $20398, ad esempio, cioe' scrivere l'indirizzo di partenza del loop, ossia dove si trovava il clr.l (a0)+, allo stesso modo, invece di scrivere LEA START,a0 avremmo dovuto scrivere lea $123456,a0, cioe' l'indirizzo da dove partire per pulire. Immaginatevi poi se avessimo inserito un istruzione in piu' nel ciclo! in questo caso START si sarebbe spostato in avanti, e avremmo dovuto riscrivere il numero esatto nel LEA $123456,a0 ossia nel LEA START,a0. Invece dando un nome a ogni PUNTO DEL PROGRAMMA, come si da il nome ad un fiume, si indica con quel nome l'indirizzo di partenza di quella cosa (E NON IL SUO CONTENUTO! SE FACCIO LEA START,A0, in A0 non va il contenuto di start! ma dove e'!). Durante l'assemblaggio l'assemblatore si occupa di sistemare tutte le label e sostituirle con gli indirizzi che rappresentano.
questo programmino fa una pulizia a partire dall'indirizzo messo in a0, fino all'indirizzo che mettiamo in a1: per verificare cio', assemblate con A e fate M START (prima di fare j) per verificare che sono stati messi in quel punto dei $fe consecutivi; come ulteriore verifica fate D Inizio, e noterete che in a0 viene messo lo stesso numero che compare accanto al primo LINE_F, ossia l'indirizzo del primo $FE che viene interpretato come LINE_F da ASMONE, mentre in a1 viene messo l'indirizzo di THEEND, ossia come potete vedere la fine dei $FEFE e l'inizio degli 00000000. Ora eseguite con il J: se fate D INIZIO potete verificare che i bytes sono stati azzerati (e ora sono interpretati come ORI.B #$0,d0), allo stesso modo con M START potete verificarlo, inoltre potete verificare che A0 ed A1 hanno lo stesso valore.
ORA VI INSEGNERO' UNA UTILITA' DELL'ASMONE MOLTO CARINA PER VERIFICARE IL

```
FUNZIONAMENTO DEI VOSTRI PROGRAMMI:
invece di fare A, provate a fare AD!!!!!
In questo modo dopo aver assemblato enterete nel DEBUGGER!!!!
CALMA E SANGUE FREDDO: vi apparira' il sorgente cosi' come lo avete scritto,
e al lato destro avrete sotto controllo tutti i registri che appaiono in
una colonna uno sotto l'altro: d0,d1,d2..a0,a1,a2.. eccetera.
Noterete che la prima linea del listato, in questo caso lea START,a0, e'
scritta in negativo: questo evidenzia che siamo a quella linea. Ora potete
controllare l'esecuzione del programma istruzione dopo istruzione verificando
cosa avviene nei registri! Nell'ultima linea in basso si vedono le istruzioni
disassemblate come col comando D una alla volta, con l'indirizzo all'estrema
sinistra, seguito dall'istruzione in formato BYTES, seguito dall'istruzione
in formato COMANDO (es. CLR.L (a0)+, che in bytes verificherete che e' $4298)
Per eseguire una istruzione alla volta e andare alla seguente basta premere
il tasto che sposta il cursore in avanti, quello con la freccia rivolta
verso destra: noterete come dopo aver eseguito la prima istruzione,
in a0 sara' messo l'indirizzo di START, mentre dopo la seconda sara'
caricato l'indirizzo di THEEND; scendendo nel loop noterete come a0
ogni volta sara' aumentato di 4 e come si ritornera' a CLELOOP
dopo il BNE fino a che a0 non avra' raggiunto il valore di a1.
Una volta raggiunto l'RTS della fine, o se volete anche prima, potete
uscire dal debugger con il tasto ESC.
Se contate le volte che viene eseguito il CLR.L (a0)+ verificherete che
viene eseguito 10 volte, infatti per pulire 40 bytes una long alla volta,
ossia 4 bytes allla volta, occorranno 10 passaggi (10*4=40).
Provate a modificare il CLR.L (a0)+ con CLR.W (a0)+ e noterete che sono
necessari 20 passaggi (infatti 20*2=40) e che ogni volta a0 e' aumentato
di 2, mentre sostituendolo con CLR.B (a0)+ serviranno 40 passaggi e il
registro a0 sara' incrementato di 1 ogni volta.
Per verifica caricate nuovamente i listati visti fino ad ora e seguiteli
passo passo con il comando AD.
NOTA: il debugger non puo' essere impiegato per tutti i programmi, perche'
quelli che disabilitano il sistema operativo disabilitano anche il debugger!

Per sincerarvi che viene pulito tutto quello che si trova tra la label START:
e la label THEEND:, sia che siano 40 bytes che 200 o altri, infatti provate
a fare questa modifica:

START:
        dcb.b        80,$fe        ; METTI QUA IN MEMORIA 80 bytes $fe.

THEEND:                ; questa label segna la fine degli 80 bytes...

Se fate gli stessi passaggi con AD noterete che vengono fatti il doppio dei
cicli, perche' la distanza tra START e  THEEND e' raddoppiata.
Infatti immaginatevi che il programma sia una strada, in cui START corrisponde
al numero civico 10... nel primo caso ci sono 40 bytes di distanza, quindi
THEEND sarebbe l'equivalente del numero civico 50 (10+40), e se l'abitante
in START deve andare a trovare l'amico che sta a THEEND deve fare 40 passi
lunghi 1 byte. Se invece START rappresenta sempre il numero 10, ma l'amico
THEEND e' andato a stare a 80 bytes di distanza, anziche' 40, si trovera'
all'indirizzo 90, e l'amico START per raggiungerlo dovra' fare questa volta
80 passi da un byte.
```

18.7 Lezione2g

```
; Lezione2G.s

Inizio:
        lea        THEEND,a0        ; metti in a0 l'indirizzo da dove iniziare
```

```
        lea        START,a1        ; metti in a1 l'indirizzo dove finire
CLELOOP:
        clr.l        -(a0)        ;aggiungi 4 ad a0 (long!), poi azzera la long
        cmp.l        a0,a1        ; a0 e' uguale ad a1? Cioe' siamo all'indirizzo START?
        bne.s        CLELOOP        ; se no, torna ad eseguire CLELOOP...
        rts                ; ESCI dal prog e torna ad ASMONE

START:
        dcb.b        40,$fe        ; METTI QUA IN MEMORIA 40 bytes $fe.
THEEND:                        ; questa label segna la fine dei 40 bytes...

        dcb.b        10,0        ; mettiamo 10 bytes azzerati qua tanto per sfizio

        end
```

questo programmino fa una pulizia a partire dall'indirizzo messo in a0, fino
all'indirizzo che mettiamo in a1: la diversita' con LEZIONE2f.s in cui e'
usato un CLR.L (a0)+ e' che qua si parte dalla fine dei bytes da pulire e
si arriva "indietreggiando" fino all'inizio.
per verificare cio', fate un AD e noterete che ogni volta che viene eseguito
il CLR.L -(a0) il registro a0 decrementa fino a raggiungere a1, cioe' START.
verificate pure con M START che dopo l'esecuzione la pulizia e' avvenuta,
e se vi aggrada modificate pure il CLR.L -(a0) con CLR.W -(a0) e noterete
che sono necessari 20 passaggi (infatti 20*2=40) e che ogni volta a0 e'
diminuito di 2, mentre sostituendolo con CLR.B -(a0) serviranno 40 passaggi
e il registro a0 sara' decrementato di 1 ogni volta.

18.8 Lezione2h

```
; Lezione2h.s

Inizio:
        lea        $dff006,a0        ; VHPOSR - metti $dff006 in a0
        lea        $dff180,a1        ; COLOR00 - metti $dff180 in a1
        lea        $bfe001,a2        ; CIAAPRA - metti $bfe001 in a2
Waitmouse:
        move.w        (a0),(a1)+        ; metti il valore di $dff006 nel color 0,
                                ; ovvero $dff180 (contenuto in a1)
                                ; e incrementa di 2 a1, portandolo a $dff182,
                                ; ossia il color 1
        move.w        (a0),-(a1)        ; decrementa di 2 a1, riportandolo a $dff180,
                                ; poi metti $dff006 nel color 0
        btst        #6,(a2)                ; tasto sinistro del mouse premuto?
        bne.s        Waitmouse        ; se no ritorna a waitmouse e ripeti
        rts                        ; esci

        END
```

Con questo ciclo si notano benissimo le differenze tra (a1)+ e -(a1), infatti
sono messi in maniera da annullarsi a vicenda: mentre il primo (a1)+
incrementa a1 di una word portandolo a $dff182, il -(a1) seguente riporta
a1 a $dff180 e scrive sempre nel color 0.
Infatti le due istruzioni si possono riscrivere semplicemente:

```
        move.w        (a0),(a1)
        move.w        (a0),(a1)
```

Verificate lo scambiarsi degli indirizzi $dff180 e $dff182 nel reg. a1 facendo
un AD.
Ricordatevi BENE che quando vedete un + DOPO una parentesi il registro

viene AUMENTATO (+!!!) DOPO l'operazione, mentre se vedete un - PRIMA di
una parentesi il registro viene DIMINUITO (-!!!) PRIMA!!!
NOTA: potete terminare il ciclo durante l'AD tenendo premuto il tasto sinistro
quando si esegue il btst; una volta raggiunto l'RTS premete ESC per tornare.

18.9 Lezione2i

```
; Lezione2i.s

Inizio:
        lea        $dff000,a0        ; metti $dff000 in a0
Waitmouse:
        move.w        6(a0),$180(a0)        ; metti il valore .w di $dff006 nel color 0
                              ; 6(a0)=$dff000+6, $180(a0)=$dff000+$180
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        Waitmouse        ; se no ritorna a waitmouse e ripeti tutto
        rts                          ; esci

        END
```

In questa variazione del primo listato sono presenti delle distanze di
indirizzamento: in a0 viene messo l'indirizzo $dff000 (in questo caso si
sceglie perche' e' pari e quando si fanno le distanze di indirizzamento
si possono riconoscere a quali indirizzi si fa riferimento: per esempio
il color0, cioe' il $dff180, si puo' raggiungere con $180(a0), ed e'
evidente che si tratta del $dff180. Se per esempio in a0 avessi messo
l'indirizzo $dff013, per indicare il color0 la giusta distanza di
indirizzamento sarebbe stata $16d(a0), infatti $dff013+$16d=$dff180).
Da notare che il registro a0 non viene mai cambiato, rimane sempre
$dff000, e ogni volta il processore calcola a quale indirizzo ci stiamo
riferendo sommando la distanza di indirizzamento all'indirizzo in a0.
In quasi tutti i programmi che usano la grafica l'indirizzo $dff000 viene
messo in qualche registro per farci la distanza di indirizzamento (o OFFSET),
infatti in questo modo si possono raggiungere tutti i registri CUSTOM
(che finiscono a $DFF1fe).
Si puo' indicare un offset al massimo da -32768 a +32767, ossia da -$8000
a $7FFF.

NOTA:
fate attenzione alla differenza che c'e' tra il LEA ed il MOVE quando
si usa una distanza di indirizzamento:

```
        MOVE.L        $100(a0),a1
```

Copia la longword CONTENUTA nell'indirizzo che si trova piu' avanti di quello
in a0 di $100 bytes, nel registro a1. QUINDI: "FAI LA SOMMA TRA L'INDIRIZZO
IN A0 E IL NUMERO PRIMA DELLA PARENTESI; IL RISULTATO E' L'INDIRIZZO DA CUI
VERRA' COPIATA LA LONGWORD IN A1".

mentre:

```
        LEA        $100(a0),a1
```

Mette in a1 l'indirizzo risultante dalla somma di a0+$100, non il suo contenuto
infatti il comando LEA serve solo per caricare INDIRIZZI, non CONTENUTI.

Facciamo un esempio per chiarire: consideriamo gli indirizzi di memoria come
gli indirizzi di una lunga strada assolata con tante villette in fila, ognuna
con un numero civico. Se mettiamo in a0 l'indirizzo 0, ossia l'indirizzo della
prima casa, con l'istruzione MOVE.L $100(a0),a1, non facciamo altro che mettere

```
in a1 il tappeto e i mobili dell'ingresso della casa n.$100, ossia ne copiamo
il CONTENUTO per la lunghezza di una longword in a1.
Invece con LEA $100(a0),a1 mettiamo in a1 l'indirizzo della casa $100 senza
entrarci. La differenza sta che con il MOVE in a1 abbiamo messo i mobili,
con il lea invece l'indirizzo. Per CONTENUTO intendo cio' che e' negli
indirizzi, infatti in ogni indirizzo (ogni casa) c'e' sempre qualcosa: puo'
essere un numero (quando ci sono i mobili) oppure puo' essere vuoto (quando la
casa e' abbandonata, ma da cui si puo' prendere ZERO ($00) comunque).

Per esempio l'istruzione

        LEA        $100(a1),a1

E' equivalente all'istruzione:

        ADD.W       #$100,a1

Perche' in a1 viene messo, appunto, l'indirizzo in a1+$100.

NOTA: le distanze di indirizzamento le potere scrivere in decimale o in
esadecimale (col simbolo del $) a piacere, e potete anche mettere delle
moltiplicazioni o delle divisioni etc:

        lea        $10*3(a1),a2        ; ovvero sara' assemblato LEA $30(a1),a2
                                ; infatti * significa MOLTIPLICA
```

18.10 Lezione2l

```
; Lezione2l.s

Inizio:
        lea        $dff000,a0        ; metti $dff000 in a0
Waitmouse:
        move.w       #$20,$1dc(a0)        ; BEAMCON0 (ECS+) Risoluzione video PAL
        bsr.s       Lampeggio        ; Fa lampeggiare lo schermo
        bsr.s       ColorFreccia        ; Fa lampeggiare la freccia
        btst       #2,$16(a0)        ; POTINP - Tasto destro del mouse premuto?
                                ; (bit 2 del $dff016
        bne.s       nonpremuto        ; Se non e' premuto, salta FaiConfusione
        bsr.s       FaiConfusione        ;
nonpremuto:
        btst       #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s       Waitmouse        ; se no ritorna a waitmouse e ripeti tutto
        rts                        ; esci

ColorFreccia:
        moveq       #-1,d1                ; OSSIA moveq #$FFFFFFFF,d1
        moveq       #20-1,d0        ; numero di cicli colorfreccia
flash:
        subq.w       #8,d1                ; cambia il colore da mettere in $dff1a4
        move.w       d1,$1a4(a0)        ; COLOR18 - metti il valore di d1 in $dff1a4
                                ; (il colore della freccia del mouse!)
        dbra       d0,flash
        rts

Lampeggio:
        move.w       6(a0),$180(a0)        ; metti il valore .w di $dff006 nel color 0
        move.b       6(a0),$182(a0)        ; metti il valore .b di $dff006 nel color 1
        rts
```

```
FaiConfusione:
        move.w        #0,$1dc(a0)        ; BEAMCON0 (ECS+) Risuluzione video NTSC
        rts

        END
```

Questo programmino e' interessante solo per la sua struttura, infatti
ha un programma principale, quello da Inizio all'RTS, il quale richiama
delle subroutine (ovvero sottoprogrammi, che non sono altro che parti del
programma denominati da una label (cioe' da un nome) e terminanti in un RTS.
Con il debugger "AD" provate a seguire il corso del programma: per seguire
tutte le subroutine andate avanti con il tasto con la freccia verso destra,
e noterete tra l'altro nella routine ColorFreccia come il registro d0
sia decrementato di 1 in 1.

Il problema fondamentale delle strutture BSR/BEQ/BNE/RTS sta nel fatto che
tutto e' regolato da salti che possono determinare un ritorno tramite RTS al
punto dove e' stato eseguito tale salto (BSR LABEL), e da salti che invece
sono come i rami di un albero: una volta scelto se prendere la diramazione
destra o sinistra si continua per quella e non si puo' piu' tornare indietro

```
                  ramo 1
                  _______ _ _ eccetera _ _ _ RTS, uscita da questa parte
    bivio beq/bne /
    -------------/
                \ ramo 2
                 \______ _ _ eccetera _ _ _ RTS, uscita da questa altra parte
```

Un salto BEQ/BNE e' come decidere di andare a Milano o a Palermo, si passa da
altre strade, e una volta arrivati alla destinazione si passa la notte in una
di quelle due citta' (dove troviamo l'RTS), avendo percorso diverse autostrade.

Invece se troviamo un BSR.w Milano, saltiamo a Milano, eseguiamo le istruzioni
che troviamo a Milano, poi quando troviamo un RTS ci "teletrasportiamo" al
punto dove avevamo imboccato la strada per Milano, miracolosamente, e' come
se leggessimo un libro magico, in cui in ogni pagina c'e' la figura di un
paesaggio, dunque con un AbraCadaBSR entriamo nel disegno della prima pagina,
ci passiamo un po' di tempo, poi imbattendoci in un AmuletRTS torniamo seduti
davanti al libro, pronti ad un AbraCadaBSR nella seconda pagina.

NOTA1: Premendo il tasto destro viene eseguita una routine che altrimenti
viene saltata:

```
        btst        #2,$16(a0)        ; Tasto destro del mouse premuto?
                                  ; (bit 2 del $dff016 - POTINP)
        bne.s        nonpremuto        ; Se non e' premuto, salta FaiConfusione
        bsr.s        FaiConfusione        ;
nonpremuto:
```

ricordatevi bene questo metodo per eseguire una subroutine solo a patto
che una certa condizione sia soddisfatta, in questo caso che il tasto
destro del mouse sia premuto; programmando si fanno spesso di queste cose.
Il registro usato per fare "Confusione" e' il $dff1dc, il cui bit 5 serve
per scambiare la modalita' video tra PAL europea o NTSC americana; questo
registro esiste solo nei computer fabbricati dopo il 1989, a qualcuno che
ha un'Amiga vecchio potrebbe non funzionare. Se vi funziona noterete che
premendo il tasto destro lo schermo oltre a lampeggiare sembrera' che
esploda, infatti scambiando molto velocemente modalita' questo e' il
risultato. Se volete fare 2 programmini richiamabili da AmigaDos che scambino
il modo video, basta che facciate:

```
        move.w        #0,$dff1dc        ; BEAMCON0
        rts
```

assemblatelo, e salvatelo su un disco con WO (cioe' come file che potete
eseguire) con il nome NTSC, poi assemblate quest'altro:

```
        move.w        #$20,$dff1dc        ; BEAMCON0
        rts
```

E salvatelo come PAL. Da SHELL potrete cosi' cambiare modo video
chiamando i 2 programmini PAL e NTSC.

Se non vi orientate in questo programma considerate che quelli VERI sono
mille volte piu' complicati come BSR vari, quindi vedete di capirlo al 100%
prima di cominciare la LEZIONE3, intitolata: "POTEVAMO STUPIRVI CON EFFETTI
SPECIALI E COLORI ULTRAVIVACI, MA NON SAPPIAMO ANCORA FARE".

18.11 Lezione2m

```
; Lezione2m.s

; Dimostrazione che lavorando con i registri indirizzi a0...a7 si opera sempre
; su tutta la longword, sia con il consueto .L che con il .W.


inizio:
        move.l        #$FFFFFF,d0
        ADDQ.W        #4,d0                ; Aggiungi.w 4 a d0, ma lavora solo sulla word
                              ; perche' siamo su un registro DATI (lo stesso
                              ; farebbe su una label)
        lea        $FFFFFF,a0
        ADDQ.W        #4,a0                ; Aggiungi.w 4 ad a0, ma lavorando su un
                              ; registro INDIRIZZI l'add coinvolge tutto
                              ; l'indirizzo, ossia la longword.
        rts

        end
```

Provate a fare un debug di questo listato (AD) e passo passo noterete la
differenza principale tra un registro indirizzi e un registro dati o qualsiasi
label. Questa differenza e' che su registri indirizzi si lavora sempre su tutto
l'indirizzo, ossia su tutta la longword, infatti non e' possibile operare con
istruzioni .B su tali registri, e quando si lavora col .W (possibile solo nel
caso che aggiungiamo/sottraiamo/muoviamo numeri piu' piccoli di una word) il
risultato e' lo stesso di un .L. Dunque potete anche sempre usare il .L, ma
nei casi in cui e' possibile conviene "OTTIMIZZARE" l'istruzione cambiandola
in .W, dato che e' piu' veloce di quella .L.
Facendo il DEBUG di questo listatino noterete che l'ADDQ.W #4,d0 opera solo
sulla word, appunto, di D0, e lo cambia in $00FF0003, dato che dopo $FFFF
riparte da capo la numerazione con $0000, poi arriva a $0003, ma non e'
coinvolta la parte alta del numero.
Se invece faceste un ADDQ.L #4,d0 (provate!) tale ADD coinvolgerebbe l'intera
LONG, trasformandola in $01000003, infatti dopo $00FFFFFF viene $01000000.
Invece operando su registri indirizzi l'ADD.W fa come l'ADD.L, solo che non
puo' essere usato sempre, ad esempio non puo' essere usato per un numero come
per esempio $123456. Nonostante non sia un errore, ricordatevi sempre di usare
il .W anziche' il .L in queste istruzioni con i registri indirizzi per fare
codice leggermente piu' veloce.

```
ADD.L       #$123,a0        ; ottimizzabile in ADD.W #$123,a0
ADD.L       #$12345,a0        ; non ottimizzabile
```

CAPITOLO 19

LEZIONE 3

19.1 Lezione3a

```
; Lezione3a.s        - COME SI ESEGUE UNA ROUTINE DEL SISTEMA OPERATIVO

Inizio:
        move.l        $4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
mouse:
        move.w        $dff006,$dff180        ; metti VHPOSR in COLOR00 (lampeggio!!)
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        4.w,a6                ; Execbase in a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        rts

        END
```

Questo e' il primo listato in cui usiamo una routine del sistema operativo!
E guarda caso proprio quella che disabilita il sistema operativo stesso!
Infatti noterete che durante l'esecuzione la freccia controllata dal mouse si
blocca, premendo il tasto destro non appaiono i menu a tendina, i disk drive
smettono di fare click. E state attenti che anche il comando AD, ossia il
debugger, che usa il sistema operativo, viene disabilitato, rimanendo bloccato!
ricordatevi quindi che quando disabilitiamo il sistema operativo, o addirittura
puntiamo una nostra copperlist, il debugger serve fino a che il sistema
operativo e' vivo!
Provate comunque a fare "AD", premendo il tasto cursore con la freccia verso
destra (con questo tasto infatti si "penetra" dentro i BSR e JSR, mentre col
tasto cursore con la freccia verso il basso si salta il debug di BSR e JSR).
Passata la prima istruzione, il MOVE.L 4.w,a6, comparira' nel registro A6
l'indirizzo che era contenuta nela long composta dai 4 bytes $4,$5,$6,$7.
Premete ESC e veririficate facendo un "M 4", premendo 4 volte return: troverete
infatti lo stesso indirizzo. Questo indirizzo e' messo in quel punto dal kick
ogni volta che si resetta o si accende l'amiga.

```
Riprendete il debug, passate il MOVE.L 4.w,a6, ed "entrate" nel JSR -$78(a6)
con i cursore: per seguire la subroutine dovete osservare la linea disassembly
in fondo allo schermo, infatti noterete una istruzione JMP $fcxxxx o $f8xxxx,
a seconda se avete un kick 1.3 o 2.0/3.0. Siete all'indirizzo che era in $4
meno $78, e vi trovate nella memoria RAM del vostro Amiga ancora, dove pero'
trovate un JMP che vi fara' saltare nella ROM. Infatti, ogni volta che
l'Amiga perde quel secondo o due durante il RESET, o l'accensione, crea in
memoria una TABELLA DI JMP, il cui indirizzo finale e' messo in $4.
Ogni JMP salta all'indirizzo di quel particolare kickstart dove si trova la
routine corrispondente alla posizione di quel JMP rispetto alla sua fine.
Infatti facendo JSR -$78(a6) si disabilita il multitasking sia su un kick 1.2
che su un kick 1.3, o 2.0 o 3.0, cosi' pure per quelli futuri.
Se per esempio nel kick 1.3 la routine nel ROM si trovasse a $fc12345, il
JMP posto a $78 bytes sotto l'indirizzo base sara' JMP $fc12345, mentre
se su un kick 2.0 la routine in questione fosse a $f812345, il JMP in questione
sara' a $f812345. Questo sistema permette anche di caricare un kickstart in
RAM: bastera' poi che faccia una TABELLA DI JMP che puntino alle sue routines.
Fermate il debug con ESC dopo esservi annotati a che indirizzo era il JMP,
e provate a fare un "D quell'indirizzo" (l'indirizzo dell'istruzione e' il
primo numero a sinistra in fondo allo schermo! oppure lo trovate anche in
fondo alla lista dei registri sulla destra, e' il registro PC, ossia Program
Counter, che registra l'indirizzo in esecuzione, basta che gli aggiungiate un
$ davanti). Verificherete che c'e una fila di JMP; questo e' un esempio:

        JMP         $00F817EA         ; -$78(a6), ossia il DISABLE
        JMP         $00F833DC         ; -$72(a6) un'altra routine
        JMP         $00F83064         ; -$6c(a6) un'altra routine...
        JMP         $00F80F74         ; ....
        JMP         $00F80F0C
        JMP         $00F81B74
        JMP         $00F81AEC
        JMP         $00F8103A
        JMP         $00F80F3C
        JMP         $00F81444
        JMP         $00F813A0
        JMP         $00F814F8
        JMP         $00F82842
        JMP         $00F812F8
        JMP         $00F812D6
        JMP         $00F80B38
        JMP         $00F82C24
        JMP         $00F82C24
        JMP         $00F82C20
        JMP         $00F82C18

Per inserire pezzi di disassemblato nel sorgente ho usato il comando "ID",
in cui bisogna dire l'inizio e la fine della zona da inserire:

BEG> qua mettete l'indirizzo o la label, provate con l'indirizzo del JMP
END> mettete l'indirizzo finale, oppure $xxxxx+$80, intendendo per $xxxxx
     l'indirizzo di partenza: in questo caso si otterra' il disassemblato
     a partire dall'indirizzo $xxxxx fino a $80 bytes dopo.

REMOVE UNUSED LABELS? (Y/N)         ; QUA METTETE UN "Y". Se non lo mettete sara'
                                ; messa una label recante l'indirizzo ad
                                ; ogni linea di codice, anziche' solo dove
                                ; serve la label. Provate a fare un "ID"
                                ; di questo listato per verificare la
                                ; differenza.

Esempio: se l'indirizzo era $32123
```

```
ID

BEG> $32123
END> $32123+$80                 ; NOTA: per riavere i vecchi indirizzi premete
                        ; il tasto cursore verso l'alto varie volte.
                        ; (Infatti premendo la freccia verso l'alto ritornano
                        ; le cose che avete scritto prima come nello SHELL)
```

e apparira', a partire dal punto dove eravate col cursore l'ultima volta nel testo, il disassemblato richiesto.

Ora vi potete immaginare quanti JSR e JMP deve eseguire il processore quando un programma gli chiede di eseguire delle routine. E tutto questo saltare fa perdere tempo, e' per questo che useremo il sistema operativo solo il minimo indispensabile.

Se continuate con il DEBUG dopo il JMP, vi ritroverete nella ROM, cioe' all'indirizzo del JMP: di solito il DISABLE e' cosi':

```
        MOVE.W        #$4000,$dff09a        ; INTENA - Ferma gli interrupt
        ADDQ.B        #1,$126(a6)        ; Ferma il sistema operativo
        RTS
```

Se ci entrate premendo la freccia verso destra le istruzioni saranno viste, ma non eseguite (per sicurezza il debug quando esegue delle subroutine fuori dal listato, ossia di solito nella ROM, le scorre e basta), infatti potete continuare e andare nel loop del mouse e noterete che la freccia del mouse si puo' muovere e i drive fanno il click, cioe' non sono state eseguite quelle 2 operazioni. Potrete passare anche dal JSR -$7e(a6) e uscire.

Provate invece a scendere usando il tasto cursore con la freccia verso il basso: questa volta passando dal JSR -$78(a6) il programma vi scappera' di mano, perche' viene eseguita (senza pero' essere mostrata).
Potrete comunque uscire con il tasto sinistro, dopodiche' dovrete premere da ESC per uscire dal DEBUG.

Provate ora a fare queste modifiche:

1) Assemblate, fate un "D inizio" e vedrete questo:

```
        MOVE.L        $0004.W,A6
```

Provate ora a togliere il .w al 4 nel listato, assemblate e ripetete il "D":

```
        MOVE.L        $00000004,A6
```

Come vedete, in questo caso sono stati usati tutti i 4 byte dell'indirizzo, mentre prima con l'opzione .w abbiamo risparmiato 2 bytes. L'opzione .w puo' essere usata su tutti gli indirizzi linghi una sola word o meno.

2) Provate a sostituire la linea

```
        JSR        -$78(a6)
```

con le linee

```
        MOVE.W        #$4000,$dff09a        ; INTENA - Ferma gli interrupt
        ADDQ.B        #1,$126(a6)        ; Ferma il sistema operativo
```

O comunque quello che trovate nella ROM dopo il JMP (senza l'RTS finale!).

Noterete che il funzionamento e' lo stesso.

```
Potete fare la stessa cosa con il JSR -$7e(a6).
```

Figura 19.1: Lezione 3a

19.2 Lezione3b

```
; Lezione3b.s        ; LA PRIMA COPPERLIST


        SECTION        PRIMOCOP,CODE        ; Questo comando fa caricare dal sistema
                                ; operativo questa parte di codice
                                ; in FAST ram, se e' libera, oppure se c'e'
                                ; solo CHIP la carica in CHIP.

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName,a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita in d0 l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema (sempre a $26 della GfxBase)
        move.l        #COPPERLIST,$dff080        ; COP1LC - Puntiamo la nostra COP
        move.w        d0,$dff088                ; COPJMP1 - Facciamo partire la COP
mouse:
```

```
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; COP1LC - Puntiamo la cop di sistema
        move.w        d0,$dff088                ; COPJMP1 - facciamo partire la cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                            ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts

GfxName:
        dc.b        "graphics.library",0,0        ; NOTA: per mettere in memoria
                                        ; dei caratteri usare sempre il dc.b
                                        ; e metterli tra "", oppure ''
                                        ; terminando con ,0


GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library



OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C        ; Questo comando fa caricare dal sistema
                                ; operativo questo segmento di dati
                                ; in CHIP RAM, obbligatoriamente
                                ; Le copperlist DEVONO essere in CHIP RAM!

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0 - Nessuna figura, solo lo sfondo
        dc.w        $180,$000        ; Color 0 NERO
        dc.w        $7f07,$FFFE        ; WAIT - Aspetta la linea $7f (127)
        dc.w        $180,$00F        ; Color 0 BLU
        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST

        end
```

Questo programma fa "puntare" una nostra COPPERLIST, e puo' essere usato per far puntare una qualsiasi COPPERLIST, quindi e' utile per fare esperimenti col COPPER. NON DOVETE SCORAGGIARVI DALL'UTILIZZO DEL SISTEMA OPERATIVO CON L'APERTURA DELLE LIBRERIE E AFFINI, IN QUANTO IN TUTTO IL CORSO TROVERETE SOLTANTO L'APERTURA DELLA GRAPHICS.LIBRARY PER RIMETTERE A POSTO LA VECCHIA COPPERLIST E POCHE ALTRE COSE, BASTERA' DUNQUE IMPARARSI QUESTE POCHE COSE.
NOTA1: Come avete gia' notato questo listato contiene il comando SECTION, che ha la funzione di decidere gli HUNK del file eseguibile che salverete con il comando WO: ogni file eseguibile dallo shell, come ad esempio lo stesso ASMONE, e' messo in memoria RAM dal sistema operativo copiandolo dal dischetto o dall'hard disk, e questa azione di copia viene effettuata a seconda degli HUNK del file in questione, che non sono altro che parti di quel file, infatti un file e' formato da uno o piu' hunk. Ogni hunk ha la sua caratteristica, in particolare quella di DOVE DEVE ESSERE CARICATO, se solo in CHIP RAM o se e' possibile metterlo anche in FAST RAM; e' necessario usare il comando SECTION se si desidera generare un file eseguibile con delle copperlist o con dei suoni, infatti questo tipo di dati deve essere caricato sempre in CHIP RAM, altrimenti, se non si specifica il _C, il file generato con il WO avra' un hunk generico che puo' essere caricato in qualsiasi parte di memoria libera, sia essa CHIP o FAST. Molte vecchie demo o addirittura delle

demo per Amiga 1200 non funzionano su Amiga con Fast Ram proprio perche' il
file ha degli hunk caricabili in qualsiasi tipo di memoria libera, e non
funziona in computer con FAST memory, in quanto il sistema operativo tende
a riempire prima la FAST RAM della preziosa CHIP RAM: evidentemente coloro
che hanno fatto quelle vecchie demo o giochi avevano l'amiga 500 di base con
512k di chip ram, senza FAST, e i programmi gli funzionavano perche' venivano
comunque caricati in CHIP, lo stesso valga per chi ha un a500+ o un a600,
infatti hanno 1MB di CHIP solamente, ma quando questi programmi sono
caricati su un computer con la FAST ram, le FIGURE, i SUONI e le COPPERLIST
sono caricate in FAST RAM ed essendo i CHIP CUSTOM in grado di accedere
solamente alla memoria CHIP fanno dei suoni casuali e il video impazzisce,
generando alle volte degli inchiodamenti del sistema.
La sintassi del comando SECTION e' la seguente: dopo la parola SECTION si deve
scrivere il nome di quella sezione, date pure un nome a piacere, dopodiche'
si scrive quale tipo di section definiamo: se CODE o DATA, ovvero se fatta
di ISTRUZIONI o di DATI, differenza che pero' non e' molto importante, infatti
si definisce sezione CODE la prima di questo listato, che ha anche delle LABEL
con dei testi (dc.b 'graphics library'); dopodiche' si decide la cosa piu'
importante: se va caricata in CHIP o se va bene anche in FAST memory: per
decidere che deve essere CARICATA per forza in CHIP basta aggiungere un _C
al DATA o al CODE, se invece non e' aggiunto niente significa che i dati o
le istruzioni nella sezione sono caricabili in qualsiasi tipo di memoria.
Alcuni esempi:

```
        SECTION FIGURE,DATA_C        ; sezione di dati da caricare in CHIP
        SECTION        LISTANOMI,DATA        ; sezione di dati caricabile in CHIP o FAST
        SECTION Program,CODE_C        ; sezione di codice da caricare in CHIP
        SECTION Program2,CODE        ; sezione di codice caricabile in CHIP o FAST
```

Mettete la prima SECTION sempre come CODE o CODE_C, iniziando ovviamente con
delle istruzioni, dopodiche' potete fare delle section DATA o DATA_C dove non
ci sono istruzioni, facciamo un esempio:

```
        SECTION        Myprogram,CODE        ; Caricabile sia in CHIP che in FAST

        move...
        move...

        SECTION        COPPER,DATA_C        ; Assemblabile solo in CHIP

        dc.w        $100,$200....        ; $0100,$0200, ma si possono togliere
                                ; gli zeri iniziali, se per esempio
                                ; dobbiamo scrivere dc.l $00000001
                                ; sara' piu' comodo scrovere dc.l 1
                                ; allo stesso modo dc.b $0a si puo'
                                ; scrivere dc.b $a, in memoria sara'
                                ; assemblato $0a.

        SECTION        MUSICA,DATA_C        ; Assemblabile solo in CHIP

        dc.b        Pavarotti.....

        SECTION        FIGURE,DATA_C        ; solo in CHIP!

        dc.l        piramidi egizie

        END
```

Si puo' fare anche tutta una unica section CODE_C, ma frammentare almeno
a pezzi di 50k i dati grafici o sonori rende il programma piu' facile da
allocare nei buchetti della memoria di un unico pezzettone da 300k o piu'.
Inoltre considerate che caricare le istruzioni in CHIP RAM e' un peccato

```
anche perche' se sono caricate in FAST RAM, specie su un Amiga con 68020+,
sono eseguite piu' velocemente, anche di 4 volte, rispetto alla mem CHIP.
Esistono anche le section di tipo BSS o BSS_C, ne parleremo quando le useremo.
NOTA2: Avrete notato anche l'uso di (PC) nell'istruzione:

        move.l        OldCop(PC),$dff080        ; COP1LC - Puntiamo la cop di sistema

Questo (PC) aggiunto dopo il nome della label non cambia la FUNZIONE del
comando, infatti se togliete il (PC) succede la stessa cosa; piuttosto serve
a cambiare la FORMA del comando, infatti provate ad assemblare e fare un
D Mouse:

        ...                         BTST        #$06,$00BFE001
        ...                         BNE.B        $xxxxxxxx
        23FA003400DFF080        MOVE.L        $xxxxxx(PC),$00DFF080
        ...                         MOVE.W        D0,$00DFF088

Noterete che il move.l Oldcop(PC),$dff080 viene assemblato come $23fa....

Provate ora a togliere il (PC), assemblate e rifate D MOUSE:

        23F900xxxxxx00DFF080        MOVE.L        $xxxxxx,$00DFF080

Questa volta l'istruzione viene assemblata in 10 bytes anziche' 8, e si
legge chiaramente dopo il $23f9, che significa MOVE.L, l'indirizzo di Oldcop,
mentre nel caso del move.l con PC il comando inizia con $23fa e si vede in $34
anziche' l'indirizzo di OldCop!!! La differenza e' che quando non c'e' il
PC l'istruzione si riferisce ad un INDIRIZZO DEFINITO, infatti e' assemblato,
mentre un'istruzione col (PC) anziche' scrivere l'indirizzo scrive la distanza
che c'e' da se alla label in questione, in questo caso $34 bytes.
Le struzioni col (PC) si dicono RELATIVE AL PC, ossia al Program Counter, che
e' il registro dove e' scritto l'indirizzo dell'istruzione in esecuzione:
quando il 68000 arriva ad eseguire il MOVE.L OLDCOP(PC), calcola l'indirizzo
in PC+$34 ed ottiene l'indirizzo di Oldcop, appunto situato $34 bytes piu'
avanti. Questo modo e' piu' veloce e le istruzioni come gia' visto sono piu'
corte, ma si possono solo usare per label non piu' lontane di 32768 (come per
il BSR), e non si possono usare tra una section e l'altra, proprio perche'
le section sono caricate in chissa' che punto e chissa' che distanza in memoria
e quindi sarebbero troppo lontane. Infatti provate ad aggiungere la linea
LEA COPPERLIST(PC),a0 all'inizio del listato e constaterete che tentando di
assembllare ASMONE vi comunica un RELATIVE MODE ERROR, mentre togliendo il (PC)
l'istruzione viene assemblata. Vi consiglio di mettere sempre il (PC) alle
label quando e' possibile:

        LEA        LABEL(PC),a0
        MOVE.L        LABEL(PC),d0
        MOVE.L        LABEL1(PC),LABEL2        ; solo la prima label puo' essere
                                        ; seguita dal PC, la seconda MAI.
        MOVE.L        #LABEL1,LABEL2                ; in questo caso infatti non
                                        ; si puo' mettere il (PC) ne' al
                                        ; primo operando ne' al secondo.

NODIFICHE: Ora potete farvi qualsiasi copperlist! cominciate cambiando i 2
colori sapendo che il formato e' questo: $0RGB, in cui cioe' contano solo 3
numeri, $RGB, con R=RED,ossia ROSSO, G=GREEN,ossia VERDE, B=BLU
Ognuno di questi 3 numeri possono andare da 0 a 15, in notazione esadecimale,
cioe' da 0 ad F (0123456789ABCDEF), e a seconda di come si mischiano questi
3 colori di base si possono formare tutti i 4096 colori dell'Amiga (16*16*16).
Per ottenere il nero serve un $000, per un bianco un $FFF, un $999 e' grigio.
Attenzione! Non si mischiano come i colori a tempera o ad olio! per esempio
per fare il giallo occorrono ROSSO+VERDE, $dd0 ad esempio, per fare un viola
si devono mischiare ROSSO+BLU, ad esempio $d0e.
```

```
Questo sistema di mix dei colori e' lo stesso che trovate nelle PREFERENCES
del WorkBench o nella palette del DPAINT, con i 3 regolatori RGB.
Una volta fatte delle prove cambiando la prima copperlist, potete creare
delle sfumature, aggiungendo dei WAIT e dei COLOR 0 ($180,xxx), simili ai
tramonti che avete visto negli sfondi di SHADOW OF THE BEAST o di altri
giochi, o alle sfumature a barre di tante demo: ora sapete come funzionano!
Sostituite con Amiga+B+C+I questa copperlist a quella nel listato, osservate
cosa visualizza e perche', e modificatela per sincerarvi che avete chiaro
tutto, oppure per fare le sfumature di sfondo per il vostro primo gioco!!!

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0 - solo sfondo
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO
        dc.w        $4907,$FFFE        ; WAIT - Aspetto la linea $49 (73)
        dc.w        $180,$001        ; COLOR0 - blu scurissimo
        dc.w        $4a07,$FFFE        ; WAIT - linea 74 ($4a)
        dc.w        $180,$002        ; COLOR0 - blu un po' piu' intenso
        dc.w        $4b07,$FFFE        ; WAIT - linea 75 ($4b)
        dc.w        $180,$003        ; COLOR0 - blu piu' chiaro
        dc.w        $4c07,$FFFE        ; WAIT - prossima linea
        dc.w        $180,$004        ; COLOR0 - blu piu' chiaro
        dc.w        $4d07,$FFFE        ; WAIT - prossima linea
        dc.w        $180,$005        ; COLOR0 - blu piu' chiaro
        dc.w        $4e07,$FFFE        ; WAIT - prossima linea
        dc.w        $180,$006        ; COLOR0 - blu a 6
        dc.w        $5007,$FFFE        ; WAIT - salto 2 linee: da $4e a $50, ossia da 78 a 80
        dc.w        $180,$007        ; COLOR0 - blu a 7
        dc.w        $5207,$FFFE        ; WAIT - sato 2 linee
        dc.w        $180,$008        ; COLOR0 - blu a 8
        dc.w        $5507,$FFFE        ; WAIT - salto 3 linee
        dc.w        $180,$009        ; COLOR0 - blu a 9
        dc.w        $5807,$FFFE        ; WAIT - salto 3 linee
        dc.w        $180,$00a        ; COLOR0 - blu a 10
        dc.w        $5b07,$FFFE        ; WAIT - salto 3 linee
        dc.w        $180,$00b        ; COLOR0 - blu a 11
        dc.w        $5e07,$FFFE        ; WAIT - salto 3 linee
        dc.w        $180,$00c        ; COLOR0 - blu a 12
        dc.w        $6207,$FFFE        ; WAIT - salto 4 linee
        dc.w        $180,$00d        ; COLOR0 - blu a 13
        dc.w        $6707,$FFFE        ; WAIT - salto 5 linee
        dc.w        $180,$00e        ; COLOR0 - blu a 14
        dc.w        $6d07,$FFFE        ; WAIT - salto 6 linee
        dc.w        $180,$00f        ; COLOR0 - blu a 15
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79
        dc.w        $180,$300        ; COLOR0 - inizio la barra rossa: rosso a 3
        dc.w        $7a07,$FFFE        ; WAIT - linea seguente
        dc.w        $180,$600        ; COLOR0 - rosso a 6
        dc.w        $7b07,$FFFE        ; WAIT -
        dc.w        $180,$900        ; COLOR0 - rosso a 9
        dc.w        $7c07,$FFFE        ; WAIT -
        dc.w        $180,$c00        ; COLOR0 - rosso a 12
        dc.w        $7d07,$FFFE
        dc.w        $180,$f00        ; rosso a 15 (al massimo)
        dc.w        $7e07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $7f07,$FFFE
        dc.w        $180,$900        ; rosso a 9
        dc.w        $8007,$FFFE
        dc.w        $180,$600        ; rosso a 6
        dc.w        $8107,$FFFE
        dc.w        $180,$300        ; rosso a 3
        dc.w        $8207,$FFFE
```

```
        dc.w        $180,$000        ; colore NERO
        dc.w        $fd07,$FFFE        ; aspetto la linea $FD
        dc.w        $180,$00a        ; blu intensita' 10
        dc.w        $fe07,$FFFE        ; linea seguente
        dc.w        $180,$00f        ; blu intensita' massima (15)
        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST
```

Riassumendo, se ad esempio alla linea $50 impostate il color0 come verde, le linee $50 e seguenti saranno verdi, fino a che non viene cambiato nuovamente il colore dopo un wait, ad esempio un wait $6007.
Un consiglio: per fare questa copperlist OVVIAMENTE non ho scritto tutte le volte dc.w $180,$... dc.w $xx07,$FFFE!!!! Basta prendersi le due istruzioni:

```
        dc.w        $xx07,$FFFE        ; WAIT
        dc.w        $180,$000        ; COLOR0
```

Selezionarle con Amiga+B, e Amiga+C, poi farne una lunga fila premendo varie volte Amiga+i:

```
        dc.w        $xx07,$FFFE        ; WAIT
        dc.w        $180,$000        ; COLOR0
        dc.w        $xx07,$FFFE        ; WAIT
        dc.w        $180,$000        ; COLOR0
        dc.w        $xx07,$FFFE        ; WAIT
        dc.w        $180,$000        ; COLOR0
        .....
```

A questo punto basta cambiare le XX del wait e il valore di $180 ogni volta, e cancellare le istruzioni di troppo con Amiga+B e Amiga+X.
NOTA: Questo si puo' fare anche tra diversi buffer di testo dell'ASMONE, se per esempio nel buffer F2 ho un listato con una copperlist che voglio modificare, basta che la selezioni normalmente con Amiga+B e Amiga+C, poi torno al mio listato, per esempio in F5, e inserisco il pezzo preso dall'altro listato con Amiga+i.

19.3 Lezione3c

```
; Lezione3c.s        ; BARRETTA CHE SCENDE FATTA CON MOVE&WAIT DEL COPPER
                ; (PER FARLA SCENDERE USATE IL TASTO DESTRO DEL MOUSE)

        SECTION        SECONDCOP,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; COP1LC - Puntiamo la nostra COP
        move.w        d0,$dff088                ; COPJMP1 - Facciamo partire la COP

mouse:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
```

Figura 19.2: Lezione 3b

```
        bne.s       mouse               ; Se non ancora, non andare avanti

        btst       #2,$dff016        ; POTINP - Tasto destro del mouse premuto?
        bne.s       Aspetta               ; Se no, non eseguire Muovicopper

        bsr.s       MuoviCopper        ; Il primo movimento sullo schermo!!!!!
                                ; Questa subroutine fa scendere il WAIT!
                                ; e viene eseguita 1 volta ogni schermata video
                                ; infatti bsr.s Muovicopper fa si che venga
                                ; eseguita la routine nominata Muovicopper,
                                ; al termine della quale, con RTS, il 68000
                                ; torna qua ad eseguire la routine Aspetta,
                                ; e cosi' via.


Aspetta:                        ; se siamo sempre alla linea $ff che abbiamo
                                ; aspettato prima, non andare avanti.

        cmpi.b       #$ff,$dff006        ; siamo a $FF ancora? se si, aspetta la linea
        beq.s       Aspetta               ; seguente ($00), altrimenti MuoviCopper viene
                                ; rieseguito. Questo problema c'e' solo per
                                ; le routine molto corte che possono essere
                                ; eseguite in meno di "una linea del pennello
                                ; elettronico", detta "linea raster": il
                                ; ciclo mouse: aspetta la linea $FF, dopodiche'
                                ; esegue MuoviCopper, ma se la esegue troppo
                                ; in fretta ci troviamo sempre alla linea $FF
                                ; e quando torniamo al mouse, alla linea $FF
                                ; ci siamo gia', e riesegue Muovicopper,
                                ; dunque la routine e' eseguita piu' di una
                                ; volta al FRAME!!! Specialmente su A4000!
                                ; questo controllo evita il problema aspettando
```

```
                                ; la linea dopo, per cui tornando al mouse:
                                ; per raggiungere la linea $ff e' necessario
                                ; il classico cinquantesimo di secondo.
                                ; NOTA: Tutti i monitor e i televisori
                                ; disegnano lo schermo alla stessa velocita',
                                ; mentre da computer a computer puo' variare
                                ; la velocita' del processore. E' per questo
                                ; che un programma temporizzato col $dff006
                                ; va alla stessa velocita' su un A500 e su
                                ; un A4000. La temporizzazione verra'
                                ; affrontata meglio in seguito, per ora
                                ; preoccupatevi di capire il copper e il
                                ; funzionamento.


        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                                ; (le librerie vanno aperte e chiuse !!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts

;
;        Questa piccola routine fa scendere il wait del copper aumentandolo,
;        infatti la prima volta che sara' eseguito cambiera' il
;
;        dc.w        $2007,$FFFE        ; aspetto la linea $20
;
;        in:
;
;        dc.w        $2107,$FFFE        ; aspetto la linea $21! (poi $22,$23 ecc.)
;
;        NOTA: una volta raggiunto il valore massimo per un byte, ossia $FF,
;              se si esegue un ulteriore ADDQ.B #1,BARRA si riparte da 0,
;             fino a ritornare a $ff e cosi' via.

Muovicopper:
        addq.b        #1,BARRA        ; WAIT 1 cambiato, la barra scende di 1 linea
        rts

; Provate a modificare questo ADDQ in SUBQ e la barretta salira'!!!!

; Provate a cambiare l'addq/subq #1,BARRA in #2 , #3 o piu' e la velocita'
; aumentera', dato che ogni FRAME il wait si spostera' di 2,3 o piu' linee.
; (se il numero e' maggiore di 8 invece di ADDQ.B bisogna usare ADD.B)


;        DATI...


GfxName:
        dc.b        "graphics.library",0,0        ; NOTA: per mettere in memoria
                                ; dei caratteri usare sempre il dc.b
                                ; e metterli tra "", oppure ''

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
```

```
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0


;        DATI GRAFICI...


        SECTION        GRAPHIC,DATA_C        ; Questo comando fa caricare dal sistema
                                ; operativo questo segmento di dati
                                ; in CHIP RAM, obbligatoriamente
                                ; Le copperlist DEVONO essere in CHIP RAM!

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0 - no bitplanes, solo sfondo.

        dc.w        $180,$004        ; COLOR0 - Inizio la cop col colore BLU SCURO

BARRA:
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79

        dc.w        $180,$600        ; COLOR0 - inizio la zona rossa: rosso a 6

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST

        end
```

Ahh! Mi ero dimenticato di mettere il (PC) a "lea GfxName,a1", ma ora c'e'.
Chi si era accorto che ci si poteva mettere ha preso una nota positiva.
In questo programma viene eseguito un movimento sincronizzato con il
pennello elettronico, infatti la barra scende fluidamente.

NOTA1: In questo listato puo' confondere la struttura del ciclo con il test
del mouse piu' il test della posizione del pennello elettronico; quello
che dovete aver chiaro e' che le routines, o subroutines che si trovano tra
il loop mouse: e quello aspetta: sono eseguite 1 volta ogni fotogramma video:
provate infatti a sostituire il bsr.s Muovicopper con la subroutine stessa,
senza l'RTS finale ovviamente:

```
mouse:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

;        bsr.s        MuoviCopper        ; Una routine eseguita ogni fotogramma
;                                ; (Per la fluidita')

        addq.b        #1,BARRA        ; WAIT 1 cambiato, la barra scende di 1 linea

Aspetta:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta la linea
                                ; seguente, altrimenti MuoviCopper viene
                                ; rieseguito
```

In questo caso il risultato non cambia perche' anziche' eseguire l'ADDQ come
subroutine la eseguiamo direttamente, e forse in questo caso e' anche piu'
comodo; ma quando le subroutine sono piu' lunghe conviene fare vari BSR per
orientarsi. Per esempio se duplicate i bsr.s Muovicopper la routine sara'
eseguita 2 volte per fotogramma, e raddoppiera' la velocita':

```
        bsr.s        MuoviCopper        ; Una routine eseguita ogni fotogramma
        bsr.s        MuoviCopper        ; Una routine eseguita ogni fotogramma
```

L'utilita' delle subroutine sta proprio nella maggior chiarezza del programma,
immaginatevi se le nostre routines da mettere tra mouse: e aspetta: fossero di
migliaia di linee! il susseguirsi delle cose apparirebbe meno chiaro. Invece
se chiamiamo per nome ogni singola routine il tutto apparira' piu' facile.

*

Per far scendere la barra basta cambiare la COPPERLIST, in particolare
in questo esempio viene cambiato i WAIT, nel suo primo byte, ossia quello
che definisce la linea verticale da attendere:

```
BARRA:
        dc.w        $2007,$FFFE        ; WAIT - aspetto la linea $20
        dc.w        $180,$600        ; COLOR0 - inizio la zona rossa: rosso a 6
```

Mettendo una label a quel byte, si puo' cambiare quel byte agendo sulla
label stessa, in questo caso BARRA.

MODIFICHE:
Provate a cambiare il colore anziche' il wait: basta mettere una label
dove volete nella copperlist e potete cambiare quello che vi pare.
Mettete barra al colore in questo modo:

```
COPPERLIST:
        dc.w        $100,$200        ; BPLCON0 - no bitplanes, solo sfondo.

        dc.w        $180,$004        ; COLOR0 - Inizio la cop col colore BLU SCURO

;;;;BARRA:                          ; ** ANNULLO LA LABEL VECCHIA coi ;;
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79

        dc.w        $180                ; COLOR0
BARRA:                                ; ** METTO LA LABEL NUOVA AL VALORE DEL COLORE.
        dc.w        $600        ; inizio la zona rossa: rosso a 6

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST
```

Otterrete una variazione dell'intensita' del rosso, infatti cambiamo il
primo byte a sinistra del colore: $0RGB, ossia il $0R, ossia il ROSSO!!!!

Provate ora ad agire sull'intera WORD del colore: cambiate la routine cosi':

```
        addq.w        #1,BARRA        ; invece di .b operiamo sulla .w
        rts
```

Provatelo e verificherete che i colori si susseguono irregolarmente, infatti
sono il frutto del numero che aumenta: $601,$602... $631,$632... generando
dei colori non ordinatamente.

NOTA: il comando dc.w mette in memoria dei bytes, delle word o delle long,
        dunque si puo' ottenere lo stesso risultato scrivendo:

```
        dc.w        $180,$600        ; Color0

        oppure:

        dc.w        $180        ; Registro Color0
        dc.w        $600        ; valore del color0
```

Non ci sono problemi di sintassi come con i MOVE.

Lezione3c2

```
; Lezione3c2.s        ; BARRETTA CHE SCENDE FATTA CON MOVE&WAIT DEL COPPER
               ; (PER FARLA SCENDERE USATE IL TASTO DESTRO DEL MOUSE)

;        Aggiunto un controllo della linea raggiunta per fermare lo scroll


        SECTION        MaremmaCop,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1         ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)         ; OpenLibrary, routine della EXEC che apre
                              ; le librerie, e da in uscita l'indirizzo
                              ; di base di quella libreria da cui fare le
                              ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                              ; di sistema
        move.l        #COPPEROZZA,$dff080        ; COP1LC - Puntiamo la nostra COP
        move.w        d0,$dff088                ; COPJMP1 - Facciamo partire la COP


mouse:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; POTINP - Tasto destro del mouse premuto?
        bne.s        Aspetta                ; Se no, non eseguire Muovicopper

        bsr.s        MuoviCopper        ; Questa subroutine fa scendere il WAIT!
                              ; e viene eseguita 1 volta ogni schermata video
Aspetta:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta la linea
                              ; seguente, altrimenti MuoviCopper viene
                              ; rieseguito

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; COP1LC - Puntiamo la cop di sistema
        move.w        d0,$dff088                ; COPJMP1 - facciamo partire la cop

        move.l        4.w,a6                ; Execbase in A6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                              ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts

;
;        Questa piccola routine fa scendere il wait del copper aumentandolo,
;        infatti la prima volta che sara' eseguito cambiera' il
;
```

```
;        dc.w        $2007,$FFFE        ; WAIT - aspetto la linea $20
;
;        in:
;
;        dc.w        $2107,$FFFE        ; WAIT - aspetto la linea $21!
;
;        e cosi' via, fino al massimo specificato, in questo caso $fc
;

MuoviCopper:
        cmpi.b        #$fc,BARRA        ; siamo arrivati alla linea $fc?
        beq.s        Finito                ; se si, siamo in fondo e non continuiamo
        addq.b        #1,BARRA        ; WAIT 1 cambiato, la barra scende di 1 linea
Finito:
        rts

;        In questo caso se BARRA: ha raggiunto il valore $fc si salta l'addq

;        P.S: per ora non si puo' raggiungere la parte finale dello
;        schermo dopo il $FF, vi spieghero' in seguito perche' e come fare.

GfxName:
        dc.b        "graphics.library",0,0        ; NOTA: per mettere in memoria
                                        ; dei caratteri usare sempre il dc.b
                                        ; e metterli tra "", oppure ''

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        MiaCoppy,DATA_C        ; Le copperlist DEVONO essere in CHIP RAM!

COPPEROZZA:
        dc.w        $100,$200        ; BPLCON0 - no bitplanes, solo sfondo.

        dc.w        $180,$004        ; COLOR0 - Inizio la cop col colore BLU SCURO

BARRA:
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79

        dc.w        $180,$600        ; COLOR0 - inizio la zona rossa: rosso a 6

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST

        end
```

Come modifica, provate a cambiare il $fc della linea

```
        cmpi.b        #$fc,BARRA
```

Mettendoci valori diversi e verificherete che la barra scende fino alla
linea che specificate.

Lezione3c3

```
; Lezione3c3.s        ; BARRETTA CHE SCENDE FATTA CON MOVE&WAIT DEL COPPER
                ; (PER FARLA SCENDERE USATE IL TASTO DESTRO DEL MOUSE)
```

```
        SECTION        SfumaCop,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; COP1LC - Puntiamo la nostra COP
        move.w        d0,$dff088                ; COPJMP1 - Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; POTINP - Tasto destro del mouse premuto?
        bne.s        Aspetta                ; Se no, non eseguire Muovicopper

        bsr.s        MuoviCopper        ; Routine temporizzata ad 1 frame

Aspetta:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta la linea
                                ; seguente, altrimenti MuoviCopper viene
                                ; rieseguito

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; COP1LC - Puntiamo la cop di sistema
        move.w        d0,$dff088                ; COPJMP1 - facciamo partire la cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                                ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts


;        Questa routine sposta in basso una barra composta da 10 wait

MuoviCopper:
        cmpi.b        #$fa,BARRA10        ; siamo arrivati alla linea $fa?
        beq.s        Finito                ; se si, siamo in fondo e non continuiamo
        addq.b        #1,BARRA        ; WAIT 1 cambiato
        addq.b        #1,BARRA2        ; WAIT 2 cambiato
        addq.b        #1,BARRA3        ; WAIT 3 cambiato
        addq.b        #1,BARRA4        ; WAIT 4 cambiato
        addq.b        #1,BARRA5        ; WAIT 5 cambiato
        addq.b        #1,BARRA6        ; WAIT 6 cambiato
        addq.b        #1,BARRA7        ; WAIT 7 cambiato
        addq.b        #1,BARRA8        ; WAIT 8 cambiato
        addq.b        #1,BARRA9        ; WAIT 9 cambiato
        addq.b        #1,BARRA10        ; WAIT 10 cambiato
Finito:
```

```
        rts

        ; Da qua mettiamo i dati...


GfxName:
        dc.b        "graphics.library",0,0        ; NOTA: per mettere in memoria
                                                  ; dei caratteri usare sempre il dc.b
                                                  ; e metterli tra "", oppure ''

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0         ; della graphics.library

OldCop:                           ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0


; Qua c'e' la COPPERLIST, fate attenzione alle label BARRA!!!!


        SECTION        CoppyMagic,DATA_C ; Le copperlist DEVONO essere in CHIP RAM!

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0 - solo colore di sfondo
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO

BARRA:
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79
        dc.w        $180,$300        ; COLOR0 - inizio la barra rossa: rosso a 3
BARRA2:
        dc.w        $7a07,$FFFE        ; WAIT - linea seguente
        dc.w        $180,$600        ; COLOR0 - rosso a 6
BARRA3:
        dc.w        $7b07,$FFFE
        dc.w        $180,$900        ; rosso a 9
BARRA4:
        dc.w        $7c07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
BARRA5:
        dc.w        $7d07,$FFFE
        dc.w        $180,$f00        ; rosso a 15 (al massimo)
BARRA6:
        dc.w        $7e07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
BARRA7:
        dc.w        $7f07,$FFFE
        dc.w        $180,$900        ; rosso a 9
BARRA8:
        dc.w        $8007,$FFFE
        dc.w        $180,$600        ; rosso a 6
BARRA9:
        dc.w        $8107,$FFFE
        dc.w        $180,$300        ; rosso a 3
BARRA10:
        dc.w        $8207,$FFFE
        dc.w        $180,$000        ; colore NERO

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST


        end
```

```
Per far scendere la barra basta cambiare la COPPERLIST, in particolare
in questo esempio vengono cambiati i vari WAIT che compongono la barra, nel
loro primo byte, ossia quello che definisce la linea verticale da attendere:

BARRA:
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79
        dc.w        $180,$300        ; COLOR0 - inizio la barra rossa: rosso a 3
BARRA2:
        dc.w        $7a07,$FFFE        ; linea seguente
        dc.w        $180,$600        ; rosso a 6
        ...

Mettendo una label a quel byte, si puo' cambiare quel byte agendo sulla
label stessa, in questo caso BARRA.

*******************************************************************************

Vi consiglio di fare molte modifiche, anche le piu' casuali, per
prendere familiarita' col COPPER: Ve ne consiglio alcune:

MODIFICA1: provate a mettere dei ; ai primi 5 ADDQ.b in questo modo:

;        addq.b        #1,BARRA        ; WAIT 1 cambiato
;        addq.b        #1,BARRA2        ; WAIT 2 cambiato
;        addq.b        #1,BARRA3        ; WAIT 3 cambiato
;        addq.b        #1,BARRA4        ; WAIT 4 cambiato
;        addq.b        #1,BARRA5        ; WAIT 5 cambiato
        addq.b        #1,BARRA6        ; WAIT 6 cambiato
        addq.b        #1,BARRA7        ; WAIT 7 cambiato
        ....

Otterrete l'effetto "CALA IL SIPARIO", infatti la discesa parte in questo modo
dalla meta' dellla barra, e, siccome l'ultimo colore vale fino a che non
viene cambiato, in questo caso l'ultimo colore prima del wait della parte
inferiore della barra che va in fondo e' ROSSO, dunque sembra che la barra si
allunghi fino in fondo allo schermo. Togliete i ; e passiamo alla modifica 2.

MODIFICA2: Per ottenere un effetto "ZOOM" modificate cosi':(usate Amiga+b+c+i)

        addq.b        #1,BARRA
        addq.b        #2,BARRA2
        addq.b        #3,BARRA3
        addq.b        #4,BARRA4
        addq.b        #5,BARRA5
        addq.b        #6,BARRA6
        addq.b        #7,BARRA7
        addq.b        #8,BARRA8
        addq.b        #8,BARRA9
        addq.b        #8,BARRA10

Avete capito come mai si espande la barra? Perche' anziche' andare in basso
insieme i wait hanno diverse "velocita'", per cui le piu' basse si distanziano
da quelle piu' alte.


MODIFICA3: Questa volta "espanderemo" la barra non verso il basso, come nel
           caso precedente, ma centralmente:

        subq.b        #5,BARRA
        subq.b        #4,BARRA2
        subq.b        #3,BARRA3
        subq.b        #2,BARRA4
```

```
        subq.b        #1,BARRA5
        addq.b        #1,BARRA6
        addq.b        #2,BARRA7
        addq.b        #3,BARRA8
        addq.b        #4,BARRA9
        addq.b        #5,BARRA10
```

Infatti abbiamo cambiato i primi 5 addq in subq, dunque la parte superiore della barra in questo caso sale invece di scendere, e sale in maniera simile a quella dello "zoom" precedente, infatti le "velocita'" sono 5,4,3,2,1, mentre i 5 addq fanno lo stesso per la parte inferiore.

Lezione3c4

```
; Lezione3c4.s        ; BARRETTA CHE SCENDE FATTA CON MOVE&WAIT DEL COPPER
               ; (PER FARLA SCENDERE USATE IL TASTO DESTRO DEL MOUSE)

;        In questo listato viene fatta scendere una vera e propria
;        barra sfumata composta di 10 wait, dunque si agisce su 10 wait!
;        La differenza con Lezione3c3.s sta nell'utilizzo di una sola label
;        BARRA anziche' di 10 label, grazie alla distanza di indirizzamento.

        SECTION        BarraRossa,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; COP1LC - Puntiamo la nostra COP
        move.w        d0,$dff088                ; COPJMP1 - Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; POTINP - Tasto destro del mouse premuto?
        bne.s        Aspetta                ; Se no, non eseguire Muovicopper

        bsr.s        MuoviCopper        ; Sempre piu' difficile

Aspetta:
        cmpi.b        #$ff,$dff006        ; VHPOSR - Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta la linea
                                ; seguente, altrimenti MuoviCopper viene
                                ; rieseguito

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; COP1LC - Puntiamo la cop di sistema
        move.w        d0,$dff088                ; COPJMP1 - facciamo partire la cop

        move.l        4.w,a6
```

```
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l       gfxbase(PC),a1        ; Base della libreria da chiudere
                                ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts


;        Questa routine sposta una barra composta di 10 wait

MuoviCopper:
        LEA        BARRA,a0        ; mettiamo in a0 l'indirizzo di BARRA:
        cmpi.b       #$fc,8*9(a0)        ; siamo arrivati alla linea $fc?
        beq.s       Finito                ; se si, siamo in fondo e non continuiamo
        addq.b       #1,(a0)                ; WAIT 1 cambiato (indiretto senza distanza)
        addq.b       #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        addq.b       #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes, infatti
        addq.b       #1,8*3(a0)        ; dc.w $xx07,$FFFE,$180,$xxx e' una long.
        addq.b       #1,8*4(a0)        ; se quindi dall'indirizzo del primo wait
        addq.b       #1,8*5(a0)        ; facciamo una distanza di indirizzamento di
        addq.b       #1,8*6(a0)        ; 8 modifichiamo il dc.w $xx07,$fffe seguente.
        addq.b       #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        addq.b       #1,8*8(a0)        ; barra rossa ogni volta per farla scendere!
        addq.b       #1,8*9(a0)        ; ultimo wait! (il BARRA10 del sorgente prec.)
Finito:
        rts        ; P.S: Con questo RTS si torna al ciclo MOUSE che aspetta
               ; per la temporizzazione.

;        NOTA: "*" significa "moltiplicato", "/" significa "diviso"

        ; dati

GfxName:
        dc.b        "graphics.library",0,0        ; NOTA: per mettere in memoria
                                ; dei caratteri usare sempre il dc.b
                                ; e metterli tra "", oppure ''

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C        ; Le copperlist DEVONO essere in CHIP RAM!

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0 - no bitplanes
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO

BARRA:
        dc.w        $7907,$FFFE        ; WAIT - aspetto la linea $79
        dc.w        $180,$300        ; COLOR0 - inizio la barra rossa: rosso a 3
        dc.w        $7a07,$FFFE        ; WAIT - linea seguente
        dc.w        $180,$600        ; COLOR0 -rosso a 6
        dc.w        $7b07,$FFFE
        dc.w        $180,$900        ; rosso a 9
        dc.w        $7c07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $7d07,$FFFE
        dc.w        $180,$f00        ; rosso a 15 (al massimo)
        dc.w        $7e07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $7f07,$FFFE
```

```
        dc.w        $180,$900       ; rosso a 9
        dc.w        $8007,$FFFE
        dc.w        $180,$600       ; rosso a 6
        dc.w        $8107,$FFFE
        dc.w        $180,$300       ; rosso a 3
        dc.w        $8207,$FFFE
        dc.w        $180,$000       ; colore NERO

        dc.w        $FFFF,$FFFE       ; FINE DELLA COPPERLIST


        end
```

Per far scendere la barra basta cambiare la COPPERLIST, in particolare
in questo esempio vengono cambiati i vari WAIT che compongono la barra, nel
loro primo byte, ossia quello che definisce la linea verticale da attendere:

```
BARRA:
        dc.w        $7907,$FFFE       ; WAIT - aspetto la linea $79
        dc.w        $180,$300       ; COLOR0 - inizio la barra rossa: rosso a 3
        dc.w        $7a07,$FFFE       ; linea seguente
        dc.w        $180,$600       ; rosso a 6
        ...
```

Mettendo una label a quel byte, si puo' cambiare quel byte agendo sulla
label stessa, in questo caso BARRA. Pero' la barra in questione e' fatta
di 9 wait+color0, quindi per "spostarla" bisogna cambiare tutti e 9 i
wait, mentre i color0 (dc.w $180,$xxx) che si trovano sotto i wait rimangono
inalterati. Per raggiungere tutti e 9 i WAIT, anziche' mettere una LABEL
a tutti, e' piu' veloce caricare l'indirizzo del primo in un registro e
cambiare gli altri facendo delle distanze di indirizzamento:

```
MuoviCopper:
        LEA         BARRA,a0
        cmpi.b        #$fc,8*9(a0)        ; controlliamo l'ultimo wait, quello che
        beq.s        Finito               ; definisce la parte inferiore della barra.
        addq.b        #1,(a0)               ; cambio BARRA:
        addq.b        #1,8(a0)        ; cambio il byte 2 long dopo BARRA:
        addq.b        #1,8*2(a0)        ; cambio il byte 4 long dopo BARRA:
        addq.b        #1,8*3(a0)        ; cambio il byte 6 long dopo...
        addq.b        #1,8*4(a0)
        addq.b        #1,8*5(a0)
        addq.b        #1,8*6(a0)
        addq.b        #1,8*7(a0)
        addq.b        #1,8*8(a0)
        addq.b        #1,8*9(a0)
Finito:
        rts
```

NOTA: Provate a fare un "D MuoviCopper", e verificherete che gli 8*2,8*3 etc.
sono assemblati come:

```
        ADDQ.B        #1,$8(A0)
        ADDQ.B        #1,$10(A0)
        ADDQ.B        #1,$18(A0)
        ADDQ.B        #1,$20(A0)
        ADDQ.B        #1,$28(A0)
```

Ossia con il risultato di 8*2 (ossia 16, ovvero $10), di 8*3 ($18)...

Come ultima modifica, provate a cambiare il $fc della linea

```
        cmpi.b        #$fc,8*9(a0)

Mettendoci valori inferiori, e verificherete che la barra scende fino alla
linea che specificate.
```

Figura 19.3: Lezione 3c4

19.4 Lezione3d

```
; Lezione3d.s        BARRETTA CHE SALE E SCENDE FATTA COL MOVE&WAIT DEL COPPER

;        In questo listato viene usata una label come FLAG, ossia come
;        segnalazione per indicare se la barretta deve andare in alto
;        o in basso. Analizzate attentamente come funziona questo
;        programma, e' il primo del corso che puo' presentare problemi
;        a livello di ciclo condizionato.


        SECTION        CiriCop,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
```

```
                        ; di sistema
        move.l      #COPPERLIST,$dff080      ; COP1LC - Puntiamo la nostra COP
        move.w      d0,$dff088               ; COPJMP1 - Facciamo partire la COP
mouse:
        cmpi.b      #$ff,$dff006       ; VHPOSR - Siamo alla linea 255?
        bne.s       mouse              ; Se non ancora, non andare avanti

        bsr.s       MuoviCopper        ; Una routine che fa scendere e salire la barra

Aspetta:
        cmpi.b      #$ff,$dff006       ; VHPOSR - Siamo alla linea 255?
        beq.s       Aspetta              ; Se si, non andare avanti, aspetta la linea
                        ; seguente, altrimenti MuoviCopper viene
                        ; rieseguito

        btst        #6,$bfe001       ; tasto sinistro del mouse premuto?
        bne.s       mouse                ; se no, torna a mouse:

        move.l      OldCop(PC),$dff080       ; COP1LC - Puntiamo la cop di sistema
        move.w      d0,$dff088               ; COPJMP1 - facciamo partire la cop

        move.l      4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l      gfxbase(PC),a1       ; Base della libreria da chiudere
                        ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts

;
;
;
;


MuoviCopper:
        LEA        BARRA,a0
        TST.B       SuGiu               ; Dobbiamo salire o scendere? se SuGiu e'
                        ; azzerata, (cioe' il TST verifica il BEQ)
                        ; allora saltiamo a VAIGIU, se invece e' a $FF
                        ; (se cioe' questo TST non e' verificato)
                        ; continuiamo salendo (facendo dei subq)
        beq.w       VAIGIU
        cmpi.b      #$82,8*9(a0)       ; siamo arrivati alla linea $82?
        beq.s       MettiGiu       ; se si, siamo in cima e dobbiamo scendere
        subq.b      #1,(a0)
        subq.b      #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        subq.b      #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        subq.b      #1,8*3(a0)
        subq.b      #1,8*4(a0)
        subq.b      #1,8*5(a0)
        subq.b      #1,8*6(a0)
        subq.b      #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        subq.b      #1,8*8(a0)        ; barra rossa ogni volta per farla salire!
        subq.b      #1,8*9(a0)
        rts

MettiGiu:
        clr.b       SuGiu               ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                       ; fara' saltare alla routine VAIGIU, e
                        ; la barra scedera'

VAIGIU:
```

```
        cmpi.b      #$fc,8*9(a0)        ; siamo arrivati alla linea $fc?
        beq.s       MettiSu               ; se si, siamo in fondo e dobbiamo risalire
        addq.b      #1,(a0)
        addq.b      #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        addq.b      #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        addq.b      #1,8*3(a0)
        addq.b      #1,8*4(a0)
        addq.b      #1,8*5(a0)
        addq.b      #1,8*6(a0)
        addq.b      #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        addq.b      #1,8*8(a0)        ; barra rossa ogni volta per farla scendere!
        addq.b      #1,8*9(a0)
        rts

MettiSu:
        move.b      #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                       ; significa che dobbiamo risalire.


;        Questo byte, indicato dalla label SuGiu, e' un FLAG, ossia una
;        bandierina (in gergo), infatti una volta e'a  $ff e un'altra e' a
;        $00, a seconda della direzione da seguire (su o giu'!). E' appunto
;        come una bandierina, che quando e' abbassata ($00) indica che dobbiamo
;        scendere e quando e' alzata ($FF) dobbiamo salire. Viene infatti
;        eseguita una comparazione della linea raggiunta per verificare se
;        siamo arrivati in cima o in fondo, e se ci siamo arrivati cambiamo
;        la direzione (con clr.b SuGiu o move.b #$ff,Sugiu)

SuGiu:
        dc.b        0,0

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0         ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C        ; Questo comando fa caricare dal sistema
                                ; operativo questo segmento di dati
                                ; in CHIP RAM, obbligatoriamente
                                ; Le copperlist DEVONO essere in CHIP RAM!

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO
        dc.w        $4907,$FFFE        ; WAIT - Aspetto la linea $49 (73)
        dc.w        $180,$001        ; COLOR0 - blu scurissimo
        dc.w        $4a07,$FFFE        ; WAIT - linea 74 ($4a)
        dc.w        $180,$002        ; blu un po' piu' intenso
        dc.w        $4b07,$FFFE        ; linea 75 ($4b)
        dc.w        $180,$003        ; blu piu' chiaro
        dc.w        $4c07,$FFFE        ; prossima linea
        dc.w        $180,$004        ; blu piu' chiaro
        dc.w        $4d07,$FFFE        ; prossima linea
        dc.w        $180,$005        ; blu piu' chiaro
        dc.w        $4e07,$FFFE        ; prossima linea
        dc.w        $180,$006        ; blu a 6
        dc.w        $5007,$FFFE        ; salto 2 linee: da $4e a $50, ossia da 78 a 80
        dc.w        $180,$007        ; blu a 7
```

```
        dc.w        $5207,$FFFE         ; sato 2 linee
        dc.w        $180,$008         ; blu a 8
        dc.w        $5507,$FFFE         ; salto 3 linee
        dc.w        $180,$009         ; blu a 9
        dc.w        $5807,$FFFE         ; salto 3 linee
        dc.w        $180,$00a         ; blu a 10
        dc.w        $5b07,$FFFE         ; salto 3 linee
        dc.w        $180,$00b         ; blu a 11
        dc.w        $5e07,$FFFE         ; salto 3 linee
        dc.w        $180,$00c         ; blu a 12
        dc.w        $6207,$FFFE         ; salto 4 linee
        dc.w        $180,$00d         ; blu a 13
        dc.w        $6707,$FFFE         ; salto 5 linee
        dc.w        $180,$00e         ; blu a 14
        dc.w        $6d07,$FFFE         ; salto 6 linee
        dc.w        $180,$00f         ; blu a 15
        dc.w        $780f,$FFFE         ; linea $78
        dc.w        $180,$000         ; colore NERO

BARRA:
        dc.w        $7907,$FFFE         ; aspetto la linea $79
        dc.w        $180,$300         ; inizio la barra rossa: rosso a 3
        dc.w        $7a07,$FFFE         ; linea seguente
        dc.w        $180,$600         ; rosso a 6
        dc.w        $7b07,$FFFE
        dc.w        $180,$900         ; rosso a 9
        dc.w        $7c07,$FFFE
        dc.w        $180,$c00         ; rosso a 12
        dc.w        $7d07,$FFFE
        dc.w        $180,$f00         ; rosso a 15 (al massimo)
        dc.w        $7e07,$FFFE
        dc.w        $180,$c00         ; rosso a 12
        dc.w        $7f07,$FFFE
        dc.w        $180,$900         ; rosso a 9
        dc.w        $8007,$FFFE
        dc.w        $180,$600         ; rosso a 6
        dc.w        $8107,$FFFE
        dc.w        $180,$300         ; rosso a 3
        dc.w        $8207,$FFFE
        dc.w        $180,$000         ; colore NERO

        dc.w        $fd07,$FFFE         ; aspetto la linea $FD
        dc.w        $180,$00a         ; blu intensita' 10
        dc.w        $fe07,$FFFE         ; linea seguente
        dc.w        $180,$00f         ; blu intensita' massima (15)
        dc.w        $FFFF,$FFFE         ; FINE DELLA COPPERLIST


        end
```

Ora la barra se ne va su e giu', tramite l'utilizzo di una label che segna se stiamo salendo o scendendo: se la label SuGiu e' azzerata vengono eseguite le istruzioni che fanno scendere la barra, se invece non e' a zero vengono eseguite le istruzioni che la fanno risalire. All'inizio la label e' a zero, quindi vengono eseguiti gli ADDQ che la fanno scendere, fino a che, una volta raggiunto il fondo, la label SuGiu viene scritta con un $FF, quindi nei cicli seguenti, quando viene fatto il TST.b SuGiu, viene eseguita invece la serie di SUBQ che la fanno risalire, fino a che non arriva in cima, a quel punto la label SuGiu viene nuovamente azzerata, quindi vengono eseguiti nuovamente gli ADDQ che la fanno scendere, eccetera.
Con questa routine si possono verificare bene gli effetti delle modifiche: Provate a mettere un ; alle istruzioni che aspettano la linea $FF col $dff006:

```
mouse:
        cmpi.b        #$ff,$dff006        ; VHPOSR
;        bne.s        mouse                ; Se non ancora, non andare avanti

        bsr.s        MuoviCopper

Aspetta:
        cmpi.b        #$ff,$dff006        ; VHPOSR
;        beq.s        Aspetta


In questo modo perdiamo la sincronizzazione col video, e la barretta va
all'impazzata, provate ad eseguirlo cosi'!!! Come avrete notato, non si
fa nemmeno in tempo a vedere il suo movimento! Specialmente se avete un
Amiga 1200 o comunque un computer piu' veloce.
Ora invece faremo andare piu' piano la barretta eseguendola 1 volta ogni
2 fotogrammi anziche' 1 volta per fotogramma: fate questa modifica:
(Togliete anche il loop "Aspetta:")

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
;        bne.s        mouse                ; Se non ancora, non andare avanti

frame:
        cmpi.b        #$fe,$dff006        ; Siamo alla linea 254? (deve rifare il giro!)
        bne.s        frame                ; Se non ancora, non andare avanti

        bsr.s        MuoviCopper

;Aspetta:        ; tolto, non c'e' piu' rischio...
;        cmpi.b        #$ff,$dff006
;        beq.s        Aspetta

In questo caso vengono persi 2 fotogrammi di tempo, infatti quando il pennello
elettronico arriva alla linea $ff, ossia 255, viene passato il primo loop,
e si entra nel loop frame:, che attende che arrivi alla linea 254!!!! per
arrivarci pero' deve arrivare in fondo, ripartire da capo e arrivare a 254,
quindi in totale si aspettano 2 fotogrammi, ossia 2 spennellate complete.
Infatti eseguendo il listato cosi' modificato si nota che la velocita' e'
dimezzata. Per farlo andare ancora piu' piano, si possono perdere 3 fotogrammi:

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti
frame:
        cmpi.b        #$fe,$dff006        ; Siamo alla linea 254? (deve rifare il giro!)
        bne.s        frame                ; Se non ancora, non andare avanti
frame2:
        cmpi.b        #$fd,$dff006        ; Siamo alla linea 253? (deve rifare il giro!)
        bne.s        frame2                ; Se non ancora, non andare avanti
        bsr.s        MuoviCopper
        ...

Con lo stesso metodo, questa volta arrivati alla linea 254 gli chiediamo
di arrivare alla linea 253, il che costa un altro intero fotogramma.

Per verificare a che linea siete arrivati, quando uscite premendo il MOUSE
provate a fare "M BARRA", e vedrete l'ultimo valore che aveva il WAIT.
```

Figura 19.4: Lezione 3d

Lezione3d2

```
; Lezione3d2.s        BARRETTA CHE SALE E SCENDE FATTA COL MOVE&WAIT DEL COPPER


;        Routine eseguita 1 volta ogni 3 fotogrammi


        SECTION        CiriCop,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; COP1LC - Puntiamo la nostra COP
        move.w        d0,$dff088                ; COPJMP1 - Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti
frame:
        cmpi.b        #$fe,$dff006        ; Siamo alla linea 254? (deve rifare il giro!)
        bne.s        frame                ; Se non ancora, non andare avanti
frame2:
```

```
        cmpi.b        #$fd,$dff006        ; Siamo alla linea 253? (deve rifare il giro!)
        bne.s         frame2               ; Se non ancora, non andare avanti

        bsr.s         MuoviCopper        ; Una routine che fa scendere e salire la barra


        btst         #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s         mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; COP1LC - Puntiamo la cop di sistema
        move.w        d0,$dff088                ; COPJMP1 - facciamo partire la cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                              ; (vanno aperte e chiuse le librerie!!!)
        jsr         -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts

;        Routine muovicopper modificata in stile con lo "ZOOM" gia' visto

MuoviCopper:
        LEA         BARRA,a0
        TST.B         SuGiu                ; Dobbiamo salire o scendere? se SuGiu e'
                              ; azzerata, (cioe' il TST verifica il BEQ)
                              ; allora saltiamo a VAIGIU, se invece e' a $FF
                              ; (se cioe' questo TST non e' verificato)
                              ; continuiamo salendo (facendo dei subq)
        beq.w         VAIGIU
        cmpi.b        #$82,8*9(a0)        ; siamo arrivati alla linea $82?
        beq.s         MettiGiu        ; se si, siamo in cima e dobbiamo scendere
;        subq.b         #1,(a0)
        subq.b        #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        subq.b        #2,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        subq.b        #3,8*3(a0)
        subq.b        #4,8*4(a0)
        subq.b        #5,8*5(a0)
        subq.b        #6,8*6(a0)
        subq.b        #7,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        subq.b        #8,8*8(a0)        ; barra rossa ogni volta per farla salire!
        subq.b        #8,8*9(a0)
        rts

MettiGiu:
        clr.b         SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                          ; fara' saltare alla routine VAIGIU, e
                              ; la barra scedera'

VAIGIU:
        cmpi.b        #$fa,8*9(a0)        ; siamo arrivati alla linea $fc?
        beq.s         MettiSu                ; se si, siamo in fondo e dobbiamo risalire
;        addq.b         #1,(a0)
        addq.b        #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        addq.b        #2,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        addq.b        #3,8*3(a0)
        addq.b        #4,8*4(a0)
        addq.b        #5,8*5(a0)
        addq.b        #6,8*6(a0)
        addq.b        #7,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        addq.b        #8,8*8(a0)        ; barra rossa ogni volta per farla scendere!
        addq.b        #8,8*9(a0)
        rts
```

```
MettiSu:
        move.b        #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                          ; significa che dobbiamo risalire.

SuGiu:
        dc.b        0,0

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0         ; della graphics.library

OldCop:                           ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C        ; Questo comando fa caricare dal sistema
                                ; operativo questo segmento di dati
                                ; in CHIP RAM, obbligatoriamente
                                ; Le copperlist DEVONO essere in CHIP RAM!

COPPERLIST:
        dc.w        $100,$200         ; BPLCON0
        dc.w        $180,$000         ; COLOR0 - Inizio la cop col colore NERO
        dc.w        $4907,$FFFE         ; WAIT - Aspetto la linea $49 (73)
        dc.w        $180,$001         ; COLOR0 - blu scurissimo
        dc.w        $4a07,$FFFE         ; WAIT - linea 74 ($4a)
        dc.w        $180,$002         ; blu un po' piu' intenso
        dc.w        $4b07,$FFFE         ; linea 75 ($4b)
        dc.w        $180,$003         ; blu piu' chiaro
        dc.w        $4c07,$FFFE         ; prossima linea
        dc.w        $180,$004         ; blu piu' chiaro
        dc.w        $4d07,$FFFE         ; prossima linea
        dc.w        $180,$005         ; blu piu' chiaro
        dc.w        $4e07,$FFFE         ; prossima linea
        dc.w        $180,$006         ; blu a 6
        dc.w        $5007,$FFFE         ; salto 2 linee: da $4e a $50, ossia da 78 a 80
        dc.w        $180,$007         ; blu a 7
        dc.w        $5207,$FFFE         ; sato 2 linee
        dc.w        $180,$008         ; blu a 8
        dc.w        $5507,$FFFE         ; salto 3 linee
        dc.w        $180,$009         ; blu a 9
        dc.w        $5807,$FFFE         ; salto 3 linee
        dc.w        $180,$00a         ; blu a 10
        dc.w        $5b07,$FFFE         ; salto 3 linee
        dc.w        $180,$00b         ; blu a 11
        dc.w        $5e07,$FFFE         ; salto 3 linee
        dc.w        $180,$00c         ; blu a 12
        dc.w        $6207,$FFFE         ; salto 4 linee
        dc.w        $180,$00d         ; blu a 13
        dc.w        $6707,$FFFE         ; salto 5 linee
        dc.w        $180,$00e         ; blu a 14
        dc.w        $6d07,$FFFE         ; salto 6 linee
        dc.w        $180,$00f         ; blu a 15
        dc.w        $780f,$FFFE         ; linea $78
        dc.w        $180,$000         ; colore NERO

BARRA:
        dc.w        $7907,$FFFE         ; aspetto la linea $79
        dc.w        $180,$300         ; inizio la barra rossa: rosso a 3
        dc.w        $7a07,$FFFE         ; linea seguente
```

```
        dc.w        $180,$600       ; rosso a 6
        dc.w        $7b07,$FFFE
        dc.w        $180,$900       ; rosso a 9
        dc.w        $7c07,$FFFE
        dc.w        $180,$c00       ; rosso a 12
        dc.w        $7d07,$FFFE
        dc.w        $180,$f00       ; rosso a 15 (al massimo)
        dc.w        $7e07,$FFFE
        dc.w        $180,$c00       ; rosso a 12
        dc.w        $7f07,$FFFE
        dc.w        $180,$900       ; rosso a 9
        dc.w        $8007,$FFFE
        dc.w        $180,$600       ; rosso a 6
        dc.w        $8107,$FFFE
        dc.w        $180,$300       ; rosso a 3
        dc.w        $8207,$FFFE
        dc.w        $180,$000       ; colore NERO

        dc.w        $fd07,$FFFE       ; aspetto la linea $FD
        dc.w        $180,$00a       ; blu intensita' 10
        dc.w        $fe07,$FFFE       ; linea seguente
        dc.w        $180,$00f       ; blu intensita' massima (15)
        dc.w        $FFFF,$FFFE       ; FINE DELLA COPPERLIST


        end
```

In questo esempio la routine Muovicopper viene eseguita 1 volta ogni 3 FRAME, ossia 1 volta ogni 3 cinquantesimi di secondo, per rallentare l'eccessiva velocita', tramite lo stratagemma dei vari cmp con il $dff006.
D'altronde il fatto che e' eseguita ogni 3 fotogrammi la rende anche meno fluida, come si nota dagli scatti che fa nella parte inferiore.

Questo e' il momento per insegnarvi qualche trucchetto del mestiere.
Se si devono fare delle modifiche a lunghe COPPERLIST, per esempio si devono cambiare tutti gli 07 in 87, per far aspettare la meta' di ogni linea anziche' l'inizio, si puo' usare il comando REPLACE dell'editor, che permette di cambiare una data stringa di caratteri con un'altra.
Per fare la modifica che ho detto, dovete posizionarvi col cursore all'inizio della COPPERLIST, dopodiche' premere insieme i tasti "AMIGA+SHIFT+R", e comparira' in altro la scritta "Search For:". Qua dovete scrivere il testo originale da cercare, in questo caso scrivete "07,$fffe" e premete return. Ora apparira' la scritta "Replace with:". Qua dovete mettere la modifica che intendere fare: ossia "87,$fffe". A questo punto il cursore andra' sul primo 07,$fffe e apparira' la scritta "Replace: (Y/N/L/G)". A questo punto si deve decidere se scambiare o no lo 07 con l'87. Se lo volete cambiare, premete la Y, se non lo volete cambiare, premete N. Fatta la scelta il cursore andra' sul prossimo 07,$fffe e ripetera' la domanda. Cambiateli pure tutti fino alla fine della copperlist, dopodiche' fermatevi con ESC per non cambiare quelli nel commento sottostante. Se premete il G alla scelta saranno scambiati tutti gli 07,$fffe, fino alla fine del testo. Pensateci bene prima di usare il G (GLOBALE), potreste cambiare qualcosa che non andava cambiato. E' meglio procedere facendo Y o N fino alla fine della zona da cambiare, dopodiche' premete ESC per terminare, oppure premete la L all'ultima modifica da fare (indica LOCALE, ossia ULTIMO CAMBIAMENTO DA FARE).

Una volta fatta questa modifica, eseguite il listato: noterete che la barretta e le altre "sfumature" hanno una scalettatura verso il centro. Questo e' proprio perche' cambiamo colore nel mezzo ($87) anziche' all'inizio del video.

Provate ora a ricambiare tutto: fate il REPLACE, dando come stringa originale "87,$ff" e come stringa nuova "$67,$ff". Noterete che la scalettatura e' piu'

a destra. Per finire, fate un altro effetto: ora avete tutti i wait cambiati
in $xx67,$fffe, ebbene, provate a cambiarli in $xx69,$fffe, ma uno si e uno
no, ossia, immettete alla domanda del replace come prima stringa "67,$ff" e
come seconda "69,$ff", dopodiche' premete una volta Y, quella dopo N, quella
dopo Y e cosi' via, uno Y e uno N.
In questo modo una volta il colore cambiera' alla linea $67 e l'altra a $69,
creando un effetto simile all'incastro dei mattoncini, provate ad eseguirlo.

L'incastro sara' simile a questo:

```
        ooooooo+++++
        oooo++++++++
        oooooo++++++
        oooo++++++++
        oooooo++++++
```

19.5 Lezione3e

```
; Lezione3e.s        Effetto di scorrimento di uno sfondo sfumato


;        Routine eseguita 1 volta ogni 3 fotogrammi


        SECTION        CiriCop,CODE        ; anche in Fast va bene

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #MIACOPPER,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti
frame:
        cmpi.b        #$fe,$dff006        ; Siamo alla linea 254? (deve rifare il giro!)
        bne.s        frame                ; Se non ancora, non andare avanti
frame2:
        cmpi.b        #$fd,$dff006        ; Siamo alla linea 253? (deve rifare il giro!)
        bne.s        frame2                ; Se non ancora, non andare avanti

        bsr.s        ScrollColors        ; Una cosiddetta RASTER BAR!

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
```

```
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                                  ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts


;        Questa routine fa scorrere i 14 colori della nostra copperlist verde
;        in modo da simulare uno scorrimento verso l'alto continuato, come
;        se attraverso una fessura vedessimo scorrere una serie illimitata
;        di barrette sfumate. In pratica sposta ogni volta i colori copiandoli,
;        partendo copiando il secondo nel primo, il terzo nel secondo ecc., come
;        se avessimo una fila di palline colorate in serie: supponete di
;        prendere la seconda e metterla al posto della prima, che vi mettete
;        in tasca, creando un "buco": proseguirete spostando tutte le palline
;        una ad una di un posto: la terza al posto della seconda, la quarta
;        al posto della terza, e cosi' via, fino a che non arrivate alla
;        quattordicesima (l'ultima) che spostate dove era la tredicesima,
;        creando il "buco" che prima si trovava al posto della prima.
;        per tappare questo buco riprendete la prima pallina dalla tasca
;        e mettetela al posto della quattordicesima (si noti l'ultima istruzione
;        che infatti e' "move.w col1,col14", ossia dopo aver fatto "scrorrere"
;        il "buco" dalla prima posizione alla quattordicesima, lo "tappiamo"
;        con la prima pallina, creando un ciclo di continita' (infinito!) come
;        lo scorrimento della catena della bicicletta:
;
;         >>>>>>>>>>>>>>>>>>>>
;        ^                     v
;         <<<<<<<<<<<<<<<<<<<<
;
;        ma senza la parte inferiore della catena: semplicemente quando un
;        anello della catena (un colore) arriva al termine (v), viene
;        copiato alla prima posizione (^), rendendo possibile il ciclo
;        infinito:
;
;         >>>>>>>>>>>>>>>>>>>>
;        ^                     v
;
;        Infatti per interrompere la routine basta eliminare una qualsiasi
;        delle istruzioni che copiano: provate ad esempio a mettere un ;
;        alla prima: (move.w col2,col1) e verificherete che scrorre una sola
;        volta, dopodiche' finiscono i colori, essendo "ROTTO UN ANELLO DELLA
;        CATENA", che non fornisce piu' il colore precedente.


Scrollcolors:
        move.w        col2,col1        ; col2 copiato in col1
        move.w        col3,col2        ; col3 copiato in col2
        move.w        col4,col3        ; col4 copiato in col3
        move.w        col5,col4        ; col5 copiato in col4
        move.w        col6,col5        ; col6 copiato in col5
        move.w        col7,col6        ; col7 copiato in col6
        move.w        col8,col7        ; col8 copiato in col7
        move.w        col9,col8        ; col9 copiato in col8
        move.w        col10,col9        ; col10 copiato in col9
        move.w        col11,col10        ; col11 copiato in col10
        move.w        col12,col11        ; col12 copiato in col11
        move.w        col13,col12        ; col13 copiato in col12
        move.w        col14,col13        ; col14 copiato in col13
        move.w        col1,col14        ; col1 copiato in col14
        rts

GfxName:
```

```
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0       ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0


;=========== Copperlist =========================


        section        cop,data_C

MIACOPPER:
        dc.w        $100,$200        ; BPLCON0 - schermo senza bitplanes, solo il
                                ; colore di sfondo $180 e' visibile.

        DC.W        $180,$000        ; COLOR0 - iniziamo col colore NERO

        dc.w        $9a07,$fffe        ; aspettiamo la linea 154 ($9a in esadecimale)
        dc.w        $180                ; REGISTRO COLOR0
col1:
        dc.w        $0f0                ; VALORE DEL COLOR 0 (che sara' modificato)
        dc.w        $9b07,$fffe ; aspettiamo la linea 155 (non sara' modificata)
        dc.w        $180                ; REGISTRO COLOR0 (non sara' modificato)
col2:
        dc.w        $0d0                ; VALORE DEL COLOR 0 (sara' modificato)
        dc.w        $9c07,$fffe        ; aspettiamo la linea 156 (non modificato,ecc.)
        dc.w        $180                ; REGISTRO COLOR0
col3:
        dc.w        $0b0                ; VALORE DEL COLOR 0
        dc.w         $9d07,$fffe        ; aspettiamo la linea 157
        dc.w        $180                ; REGISTRO COLOR0
col4:
        dc.w        $090                ; VALORE DEL COLOR 0
        dc.w        $9e07,$fffe        ; aspettiamo la linea 158
        dc.w        $180                ; REGISTRO COLOR0
col5:
        dc.w        $070                ; VALORE DEL COLOR 0
        dc.w        $9f07,$fffe        ; aspettiamo la linea 159
        dc.w        $180                ; REGISTRO COLOR0
col6:
        dc.w        $050                ; VALORE DEL COLOR 0
        dc.w        $a007,$fffe        ; aspettiamo la linea 160
        dc.w        $180                ; REGISTRO COLOR0
col7:
        dc.w        $030                ; VALORE DEL COLOR 0
        dc.w        $a107,$fffe        ; aspettiamo la linea 161
        dc.w        $180                ; color0... (ora avete capito i commenti,
col8:                                ; posso anche smettere di metterli da qua!)
        dc.w        $030
        dc.w        $a207,$fffe        ; linea 162
        dc.w        $180
col9:
        dc.w        $050
        dc.w        $a307,$fffe        ;  linea 163
        dc.w        $180
col10:
        dc.w        $070
        dc.w        $a407,$fffe        ;  linea 164
        dc.w        $180
```

```
col11:
        dc.w        $090
        dc.w        $a507,$fffe        ;  linea 165
        dc.w        $180
col12:
        dc.w        $0b0
        dc.w        $a607,$fffe        ;  linea 166
        dc.w        $180
col13:
        dc.w        $0d0
        dc.w        $a707,$fffe        ;  linea 167
        dc.w        $180
col14:
        dc.w        $0f0
        dc.w         $a807,$fffe        ;  linea 168

        dc.w         $180,$0000        ; Decidiamo il colore NERO per la parte
                                ; di schermo sotto l'effetto

        DC.W    $FFFF,$FFFE        ; Fine della Copperlist

        END
```

MODIFICHE: Provate ad aggiungere questo comando alla fine della routine "Scrollcolors", ed otterrete un cambiamento dei colori (aggiungiamo 1 alla componente RED, ossia ROSSO)

```
        add.w   #$100,col13
```

Provate poi a cambiare il valore dell'add, per ottenere variazioni di colore diverse. Chiaramente e' un sistema un po' approssimativo per fare sfumature, ma puo' essere utile per sincerarsi di aver capito la routine.

19.6 Lezione3f

```
; Lezione3f.s                  BARRETTA SOTTO LA LINEA $FF

;        Questo listato e' identico al Lezione3d.s, fatta eccezione per
;        il fatto che la barretta si trova sotto la linea $FF che non
;        abbiamo mai oltrepassato.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
```

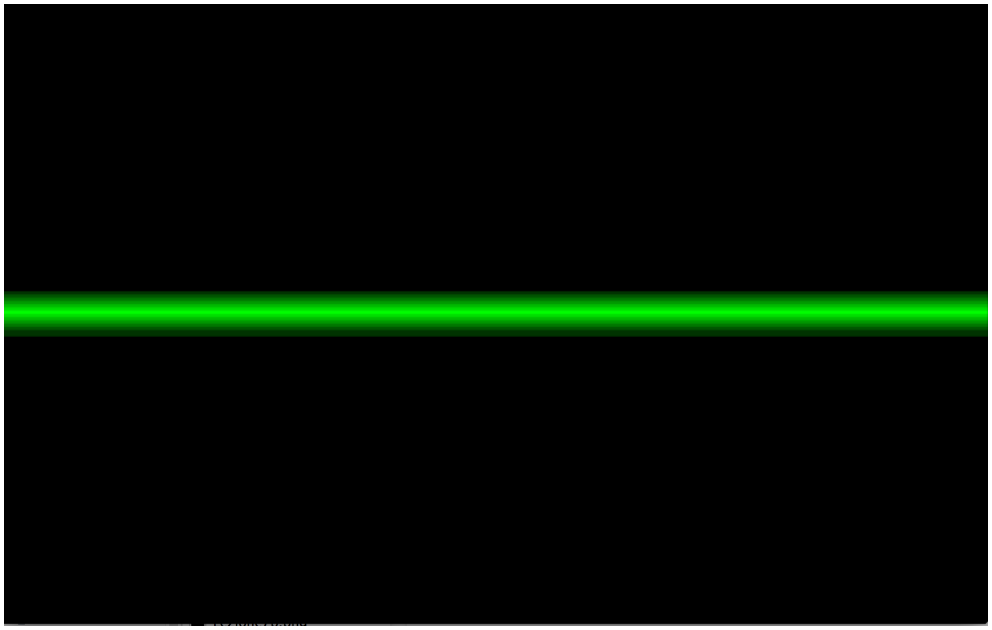


Figura 19.5: Lezione 3e

```
        bne.s        mouse                ; Se non ancora, non andare avanti

        bsr.s        MuoviCopper        ; Routine che sfrutta il mascheramento del WAIT

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta la linea
                                ; seguente, altrimenti MuoviCopper viene
                                ; rieseguito

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                                ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts

; La routine MuoviCopper e' la stessa, sono cambiati solo i valori della
; massima altezza raggiungibile, ossia $0a e del fondo dello schermo, $2c.

MuoviCopper:
        LEA        BARRA,a0
        TST.B        SuGiu                ; Dobbiamo salire o scendere? se SuGiu e'
                                ; azzerata, (cioe' il TST verifica il BEQ)
                                ; allora saltiamo a VAIGIU, se invece e' a $FF
                                ; (se cioe' questo TST non e' verificato)
```

```
                              ; continuiamo salendo (facendo dei subq)
        beq.w       VAIGIU
        cmpi.b       #$0a,(a0)        ; siamo arrivati alla linea $0a+$ff? (265)
        beq.s       MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        subq.b       #1,(a0)
        subq.b       #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        subq.b       #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        subq.b       #1,8*3(a0)
        subq.b       #1,8*4(a0)
        subq.b       #1,8*5(a0)
        subq.b       #1,8*6(a0)
        subq.b       #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        subq.b       #1,8*8(a0)        ; barra rossa ogni volta per farla salire!
        subq.b       #1,8*9(a0)
        rts

MettiGiu:
        clr.b       SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                           ; fara' saltare alla routine VAIGIU, e
                                  ; la barra scedera'

VAIGIU:
        cmpi.b       #$2c,8*9(a0)        ; siamo arrivati alla linea $2c?
        beq.s       MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        addq.b       #1,(a0)
        addq.b       #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        addq.b       #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        addq.b       #1,8*3(a0)
        addq.b       #1,8*4(a0)
        addq.b       #1,8*5(a0)
        addq.b       #1,8*6(a0)
        addq.b       #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        addq.b       #1,8*8(a0)        ; barra rossa ogni volta per farla scendere!
        addq.b       #1,8*9(a0)
        rts

MettiSu:
        move.b       #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                          ; significa che dobbiamo risalire.

;        Questo byte, indicato dalla label SuGiu, e' un FLAG, ossia una
;        bandierina (in gergo), infatti una volta e'a  $ff e un'altra e' a
;        $00, a seconda della direzione da seguire (su o giu'!). E' appunto
;        come una bandierina, che quando e' abbassata ($00) indica che dobbiamo
;        scendere e quando e' alzata ($FF) dobbiamo salire. Viene infatti
;        eseguita una comparazione della linea raggiunta per verificare se
;        siamo arrivati in cima o in fondo, e se ci siamo arrivati cambiamo
;        la direzione (con clr.b SuGiu o move.b #$ff,Sugiu)

SuGiu:
        dc.b        0,0

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                          ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w        $100,$200        ; BPLCON0
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO

        dc.w        $2c07,$FFFE        ; WAIT - una piccola barretta fissa verde
        dc.w        $180,$010        ; COLOR0
        dc.w        $2d07,$FFFE        ; WAIT
        dc.w        $180,$020        ; COLOR0
        dc.w        $2e07,$FFFE
        dc.w        $180,$030
        dc.w        $2f07,$FFFE
        dc.w        $180,$040
        dc.w        $3007,$FFFE
        dc.w        $180,$030
        dc.w        $3107,$FFFE
        dc.w        $180,$020
        dc.w        $3207,$FFFE
        dc.w        $180,$010
        dc.w        $3307,$FFFE
        dc.w        $180,$000

        dc.w        $ffdf,$fffe        ; ATTENZIONE! WAIT ALLA FINE LINEA $FF!
                                ; i wait dopo questo sono sotto la linea
                                ; $FF e ripartono da $00!!

        dc.w        $0107,$FFFE        ; una barretta fissa verde SOTTO la linea $FF!
        dc.w        $180,$010
        dc.w        $0207,$FFFE
        dc.w        $180,$020
        dc.w        $0307,$FFFE
        dc.w        $180,$030
        dc.w        $0407,$FFFE
        dc.w        $180,$040
        dc.w        $0507,$FFFE
        dc.w        $180,$030
        dc.w        $0607,$FFFE
        dc.w        $180,$020
        dc.w        $0707,$FFFE
        dc.w        $180,$010
        dc.w        $0807,$FFFE
        dc.w        $180,$000

BARRA:
        dc.w        $0907,$FFFE        ; aspetto la linea $79
        dc.w        $180,$300        ; inizio la barra rossa: rosso a 3
        dc.w        $0a07,$FFFE        ; linea seguente
        dc.w        $180,$600        ; rosso a 6
        dc.w        $0b07,$FFFE
        dc.w        $180,$900        ; rosso a 9
        dc.w        $0c07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $0d07,$FFFE
        dc.w        $180,$f00        ; rosso a 15 (al massimo)
        dc.w        $0e07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $0f07,$FFFE
        dc.w        $180,$900        ; rosso a 9
        dc.w        $1007,$FFFE
        dc.w        $180,$600        ; rosso a 6
        dc.w        $1107,$FFFE
        dc.w        $180,$300        ; rosso a 3
```

```
        dc.w        $1207,$FFFE
        dc.w        $180,$000        ; colore NERO

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST


        end
```

MIRACOLO! Abbiamo messo delle barre colorate sotto la flamigerata linea $FF!
E basta solo mettere il comando:

```
        dc.w        $ffdf,$fffe
```

E ripartire da $0107,$fffe per waitare nella parte bassa dello screen.
Questo perche' come sapete un byte contiene solo 255 valori, ossia fino
a $FF, dunque per aspettare una linea superiore a $ff basta arrivarci
con $FFdf,$FFFE, poi la numerazione riparte da 0, fino a dove arriva lo
schermo visibile, verso il $30. da notare che lo standard televisivo americano
NTSC arriva fino alla linea $FF solamente, o poco piu' in overscan, quindi
gli americani non vedono la parte bassa dello schermo sul televisore, ma a
noi non importa, perche' l'Amiga e' diffuso soprattutto in Europa dove c'e'
lo standard PAL, infatti le demo e i giochi sono quasi sempre in PAL. In certi
casi i programmatori fanno delle versioni NTSC del gioco esclusivamente per
la distribuzione in USA.

NOTA: Per ora abbiamo potuto aspettare con il $DFF006 solo una linea compresa
da $01 a $FF; spieghero' in seguito come si fa ad aspettare col $dffxxx una
linea dopo il $FF correttamente.

Figura 19.6: Lezione 3f

19.7 Lezione3g

```
; Lezione3g.s        SCORRIMENTO A DESTRA E SINISTRA TRAMITE IL WAIT del COPPER


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        bsr.w        CopperDestSin        ; Routine di scorrimento destra/sinistra

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta la linea
                                ; seguente, altrimenti MuoviCopper viene
                                ; rieseguito

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
                                ; (vanno aperte e chiuse le librerie!!!)
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts


; Questa routine anziche' agire sul primo byte a sinistra del wait, ossia
; quello che determina la posizione Y, facendo abbassare o alzare i wait con
; i colori seguenti, agisce sul secondo byte, quello delle X, generando uno
; spostamento a destra e a sinistra, regolato da 2 flags simili al SuGiu che
; abbiamo gia' visto, in questo caso si chiamano DestraFlag e SinistraFlag,
; dove sta il numero di volte che la routine VAIDESTRA o VAISINISTRA sono state
; eseguite, per limitare lo spostamento (ossia per decidere quanto andare
; avanti prima di ritornare indietro): infatti ogni volta che la routine
; VAIDESTRA viene eseguita, la "barra grigia" avanza a destra, dunque dobbiamo
; farla fermare quando raggiunge il bordo opposto dello schermo, in questo caso
; quando e' stata eseguita 85 volte, dopodiche' la facciamo tornare indietro
; eseguendo altre 85 volte la routine VAISINISTRA, che la riporta alla
; posizione iniziale, e il ciclo riparte per continuare fino a che non premiamo
; il tasto del mouse.
```

```
; DA NOTARE CHE QUESTA ROUTINE O VA A VAIDESTRA O A VAISINISTRA, NON VENGONO
; ESEGUITE TUTTE E DUE: SE VIENE ESEGUITA VAIDESTRA POI SI TORNA DA QUELLA
; ROUTINE AL LOOP MOUSE:, LO STESSO PER VAISINISTRA. SE IL CICLO VAIDESTRA E
; VAISINISTRA E' FINITO (DOPO 2*85 FRAMES) SI TORNA AL CICLO "MOUSE" DALL'RTS
; DELLA ROUTINE CopperDESTSIN direttamente, dopo aver azzerato i 2 flag.


CopperDESTSIN:
        CMPI.W        #85,DestraFlag                ; VAIDESTRA eseguita 85 volte?
        BNE.S        VAIDESTRA                ; se non ancora, rieseguila
                                        ; se e' stata eseguita gia' 85
                                        ; volte invece continua di seguito

        CMPI.W        #85,SinistraFlag        ; VAISINISTRA eseguita 85 volte?
        BNE.S        VAISINISTRA                ; se non ancora, rieseguila

        CLR.W        DestraFlag        ; la routine VAISINISTRA e' stata eseguita
        CLR.W        SinistraFlag        ; 85 volte, dunque a questo punto la barra
                                ; grigia e' tornata indietro e il ciclo
                                ; destra-sinistra e' finito, dunque azzeriamo
                                ; i due flag e usciamo: al prossimo FRAME
                                ; verra' rieseguita VAIDESTRA, dopo 85 frame
                                ; vaisinistra 85 volte per 85 frame, eccetera.
        RTS                        ; TORNIAMO AL LOOP mouse


VAIDESTRA:                        ; questa routine sposta la barra verso DESTRA
        addq.b        #2,CopBar        ; aggiungiamo 2 alla coordinata X del wait
        addq.w        #1,DestraFlag        ; segnamo che abbiamo eseguito un'altra volta
                                ; VAIDESTRA: in DestraFlag sta il numero
                                ; di volte che abbiamo eseguito VAIDESTRA.
        RTS                        ; TORNIAMO AL LOOP mouse


VAISINISTRA:                        ; questa routine sposta la barra verso SINISTRA
        subq.b        #2,CopBar        ; sottraiamo 2 alla coordinata X del wait
        addq.w        #1,SinistraFlag ; Aggiungiamo 1 al numero di volte che e'
                                ; stata eseguita VAISINISTRA.
        RTS                        ; TORNIAMO AL LOOP mouse


DestraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w        0        ; che e' stata eseguita VAIDESTRA

SinistraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w    0        ; che e' stata eseguita VAISINISTRA


;        dati per salvare la copperlist di sistema.

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
```

```
        dc.w        $100,$200        ; BPLCON0
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO


        dc.w        $9007,$fffe        ; aspettiamo l'inizio della linea $90
        dc.w        $180,$AAA        ; COLORE grigio

; Qua abbiamo "SPEZZATO" la prima WORD del WAIT $9031 in 2 bytes per poter
; mettere una label (CopBar) ad indicare il secondo byte, ossia $31 (LA XX)

        dc.b        $90                ; POSIZIONE YY del WAIT (primo byte del WAIT)
CopBar:
        dc.b        $31                ; POSIZIONE XX del WAIT (Che cambiamo!!!)
        dc.w        $fffe                ; wait - (sara' $9033,$FFFE - $9035,$FFFE....)

        dc.w        $180,$700        ; colore ROSSO, che partira' da posizioni
                                ; sempre piu' verso destra, preceduto dal
                                ; grigio che avanzera' di conseguenza.
        dc.w        $9107,$fffe        ; wait che non cambiamo (Inizio linea $91)
        dc.w        $180,$000        ; che serve a cambiare il colore in NERO
                                ; alla linea successiva alla barretta.

;        Come notate per la linea $90 servono 2 wait, uno per aspettare l'inizio
;        della linea (07) e uno, quello che modifichiamo (31), per definire in
;        che punto della linea cambiare colore, ossia passare dal giallo che e'
;        presente dalla posizione 07, al rosso che parte dopo la posizione
;        assunta dal wait che cambiamo.
;
        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST

        end
```

Bello Eh? Un effetto del genere viene usato spesso per fare gli equalizzatori a barre della musica. Lo spostamento orizzontale tramite il wait pero' ha dei limiti, infatti si possono dare solo valori dispari, e' per questo che di solito aspettiamo la linea yy07,$fffe e non yy08,$fffe. Di conseguenza si puo' scorrere a scatti di 2 pixel alla volta minimo: 7,9,$b,$d,$f,$11,$13.... oppure ogni 4 pixel, oppure 8, mantenendo comunque il numero dispari, o si rischia di far esplodere l'Amiga. Nota: il massimo valore di XX e' $e1. Come modifiche quindi posso solo consigliarvi di far aggiungere 4 o 8 anziche' 2 per cambiare velocita', in questo caso ricordatevi anche di modificare il numero massimo di volte che eseguite la routine:

```
        CMPI.W        #85/2,DestraFlag        ; 85 volte /2, ossia "diviso 2"
        BNE.S        VAIDESTRA
        CMPI.W        #85/2,SinistraFlag        ; 85/2, ossia 42 volte
        BNE.S        VAISINISTRA                ; se non ancora, rieseguila
        ....

        addq.b        #4,(a0)                ; aggiungiamo 4....
        ....
```

Oppure per un addq.b #8,a0:

```
        CMPI.W        #85/4,DestraFlag        ; 85 volte /4, ossia 21
```

se siete dei sadici provate a mettere un addq.b #1,(a0), creando dei wait XX anche pari.... nel migliore dei casi vi sparira' lo schermo a "flash" quando avviene la disparita' (infatti lo schermo si "spenge" quando un programmatore sprovveduto mette un wait con XX pari), oppure se si waita un valore strano

```
alle volte si puo' generare proprio un blocco totale del computer, una specie
di "GURU MEDITATION" del Copper. Fate dunque attenzione!!!!
In particolare posso segnalarvi alcune coordinate pari particolari che
anziche' limitarsi a far sparire lo schermo mandano proprio nel pallone il
copper, costringendovi a resettare. (almeno sull'Amiga 1200 dove le ho provate)

        dc.w        $79DC,$FFFE        ; $dc = 220! pari e particolarmente ACIDO!
                                ; fa impazzire il copper, ma non blocca
                                ; il 68000, infatti potete continuare a
                                ; lavorare "alla cieca", senza vedere nulla

        dc.w        $0100,$FFFE        ; questo invece BLOCCA tutto, non si puo'
                                ; nemmeno uscire dal programma, bisogna
                                ; resettare

        dc.w        $0300,$FFFE        ; Altro blocco totale...


Questi "ERRORI" possono essere utili in caso vogliate proteggere dei programmi:
nel caso il disco sia copiato male o la password non sia data giusta se
si fa puntare immediatamente una copperlist con questi wait indiavolati
si BLOCCA il computer peggio che con un guru del 68000, e ogni Action Replay
o altre cartucce sono disabilitate e inutilizzabili. Oppure si potrebbero
usare come autodistruzione, chissa' se mettendo tanti errori in fila si
puo' danneggiare il computer FISICAMENTE???

NOTA: Potete ottenere un effetto come questo modificando l'esempio Lezione3c.s
      che sposta in basso un wait semplicemente modificando la routine:


MuoviCopper:
        cmpi.b        #$fc,BARRA        ; siamo arrivati alla linea $fc?
        beq.s        Finito                ; se si, siamo in fondo e non continuiamo
        addq.b        #1,BARRA        ; WAIT 1 cambiato, la barra scende di 1 linea
Finito:
        rts

In questo modo, facendogli cambiare la posizione XX anziche' YY (BARRA+1), e
facendolo avanzare di 2 anziche' di 1 alla volta (numeri DISPARI!), senza
dimenticarsi che il valore massimo e' $e1, da sostituire al $fc

MuoviCopper:
        cmpi.b        #$e1,BARRA+1        ; siamo arrivati alla colonna $fc?
        beq.s        Finito                ; se si, siamo in fondo e non continuiamo
        addq.b        #2,BARRA+1        ; WAIT 1 cambiato, la barra avanza di 2
Finito:
        rts

Vedrete la prima linea spostarsi verso destra anziche' abbasarsi. Per
evidenziare l'effetto potete "ISOLARE" la linea $79 facendo diventare blu
scuro lo schermo dalla linea seguente, ossia la $7a aggiungendo queste 2
linee prima della fine della copperlist:

        dc.w        $7a07,$FFFE        ; aspetto la linea $79
        dc.w        $180,$004        ; inizio la zona rossa: rosso a 6

Nella lezione3g la difficolta' forse risiede piu' nella routine che fa andare
avanti e indietro la barra piuttosto che nel fatto che operiamo sulla posizione
XX anziche' su quella YY. In effetti le ultime lezioni che avete affrontato
hanno delle routines 68000 non troppo semplici, che sono pero' indispensabili
per generare gli effetti col copper, dunque per capire il copper stesso; nella
Lezione 4 invece le routines 68000 saranno anche piu' semplici di quelle
```

```
di questa lezione 3, dovendo spiegare come visualizzare immagini statiche.
Se non riuscite a comprendere a fondo il funzionamento delle routine delle
ultime lezioni quindi procedete con la Lezione4, e riprovate a comprenderle
quando vi troverete piu' avanti nel corso, momento in cui avrete certamente
piu' familiarita' con le routines. La Lezione3h.s e' un ampliamento della
Lezione3g.s.
```

Figura 19.7: Lezione 3g

19.8 Lezione3h

```
; Lezione3h.s        SCORRIMENTO A DESTRA E SINISTRA TRAMITE IL WAIT del COPPER

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary, routine della EXEC che apre
                                ; le librerie, e da in uscita l'indirizzo
                                ; di base di quella libreria da cui fare le
                                ; distanze di indirizzamento (Offset)
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist
                                ; di sistema
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
```

```
        bne.s       mouse               ; Se non ancora, non andare avanti

        bsr.w       CopperDestSin       ; Routine di scorrimento destra/sinistra

Aspetta:
        cmpi.b       #$ff,$dff006       ; Siamo alla linea 255?
        beq.s       Aspetta               ; Se si, non andare avanti, aspetta la linea
                            ; seguente, altrimenti MuoviCopper viene
                            ; rieseguito

        btst       #6,$bfe001       ; tasto sinistro del mouse premuto?
        bne.s       mouse               ; se no, torna a mouse:

        move.l       OldCop(PC),$dff080       ; Puntiamo la cop di sistema
        move.w       d0,$dff088               ; facciamo partire la cop

        move.l       4.w,a6
        jsr       -$7e(a6)       ; Enable - riabilita il Multitasking
        move.l       gfxbase(PC),a1       ; Base della libreria da chiudere
                            ; (vanno aperte e chiuse le librerie!!!)
        jsr       -$19e(a6)       ; Closelibrary - chiudo la graphics lib
        rts

; La routine e' come in LEZIONE3g.s, l'unica differenza e' che si agisce su
; 29 wait anziche' 1 tramite un loop DBRA che cambia un wait, salta al wait
; successivo, cambia il wait, salta al wait successivo, eccetera.

CopperDESTSIN:
        CMPI.W       #85,DestraFlag               ; VAIDESTRA eseguita 85 volte?
        BNE.S       VAIDESTRA               ; se non ancora, rieseguila
                                    ; se e' stata eseguita gia' 85
                                    ; volte invece continua di seguito

        CMPI.W       #85,SinistraFlag       ; VAISINISTRA eseguita 85 volte?
        BNE.S       VAISINISTRA               ; se non ancora, rieseguila

        CLR.W       DestraFlag       ; la routine VAISINISTRA e' stata eseguita
        CLR.W       SinistraFlag       ; 85 volte, dunque a questo punto la barra
                            ; grigia e' tornata indietro e il ciclo
                            ; destra-sinistra e' finito, dunque azzeriamo
                            ; i due flag e usciamo: al prossimo FRAME
                            ; verra' rieseguita VAIDESTRA, dopo 85 frame
                            ; vaisinistra 85 volte per 85 frame, eccetera.
        RTS                        ; TORNIAMO AL LOOP mouse


VAIDESTRA:                      ; questa routine sposta la barra verso DESTRA
        lea       CopBar+1,A0       ; Mettiamo in A0 l'indirizzo del primo valore
                            ; XX del primo wait, che si trova appunto
                            ; 1 byte dopo CopBar

        move.w       #29-1,D2       ; dobbiamo cambiare 29 wait (usiamo un DBRA)
DestraLoop:
        addq.b       #2,(a0)               ; aggiungiamo 2 alla coordinata X del wait
        ADD.W       #16,a0               ; andiamo al prossimo wait da cambiare
        dbra       D2,DestraLoop       ; ciclo eseguito d2 volte
        addq.w       #1,DestraFlag       ; segnamo che abbiamo eseguito un'altra volta
                            ; VAIDESTRA: in DestraFlag sta il numero
                            ; di volte che abbiamo eseguito VAIDESTRA.
        RTS                        ; TORNIAMO AL LOOP mouse
```

```
VAISINISTRA:                        ; questa routine sposta la barra verso SINISTRA
        lea        CopBar+1,A0
        move.w       #29-1,D2        ; dobbiamo cambiare 29 wait
SinistraLoop:
        subq.b       #2,(a0)               ; sottraiamo 2 alla coordinata X del wait
        ADD.W       #16,a0                ; andiamo al prossimo wait da cambiare
        dbra       D2,SinistraLoop        ; ciclo eseguito d2 volte
        addq.w       #1,SinistraFlag ; Aggiungiamo 1 al numero di volte che e'
                           ; stata eseguita VAISINISTRA.
        RTS                        ; TORNIAMO AL LOOP mouse

; Fate attenzione ad una cosa: cambiamo 1 wait ogni 2 soltanto, non tutti
; i wait. Ne cambiamo solo la meta' perche', a differenza di quando facciamo
; scorrere una barra in alto e basso, in cui basta 1 wait per linea
;
;        dc.w        $YY07,$FFFE        ; wait linea YY, inizio linea (07)
;        dc.w        $180,$0RGB        ; colore
;        dc.w        $YY07,$FFFE        ; wait linea YY, inizio linea (07)
;        ...
;
; In questo caso dobbiamo mettere 2 wait per ogni linea, ossia uno all'inizio
; della linea e un'altro che scorra a destra e sinistra su quella linea:
;
;        dc.w        $YY07,$FFFE        ; wait linea YY, inizio linea (07)
;        dc.w        $180,$0RGB        ; colore GRIGIO
;        dc.w        $YYXX,$FFFE        ; wait linea YY, alla posizione orizzontale
;                                ; che decidiamo noi, facendo avanzare il
;                                ; GRIGIO sul ROSSO.
;        dc.w        $180,$0RGB        ; ROSSO
;


DestraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w        0        ; che e' stata eseguita VAIDESTRA

SinistraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w    0        ; che e' stata eseguita VAISINISTRA


;        dati per salvare la copperlist di sistema.

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $100,$200        ; BPLCON0
        dc.w        $180,$000        ; COLOR0 - Inizio la cop col colore NERO

        dc.w        $2c07,$FFFE        ; WAIT - una piccola barretta fissa verde
        dc.w        $180,$010        ; COLOR0
        dc.w        $2d07,$FFFE        ; WAIT
        dc.w        $180,$020        ; COLOR0
        dc.w        $2e07,$FFFE        ; WAIT
        dc.w        $180,$030        ; COLOR0
```

```
        dc.w        $2f07,$FFFE         ; WAIT
        dc.w        $180,$040         ; COLOR0
        dc.w        $3007,$FFFE
        dc.w        $180,$030
        dc.w        $3107,$FFFE
        dc.w        $180,$020
        dc.w        $3207,$FFFE
        dc.w        $180,$010
        dc.w        $3307,$FFFE
        dc.w        $180,$000


        dc.w        $9007,$fffe         ; aspettiamo l'inizio della linea
        dc.w        $180,$000         ; grigio al minimo, ossia NERO!!!
CopBar:
        dc.w        $9031,$fffe         ; wait che cambiamo ($9033,$9035,$9037...)
        dc.w        $180,$100         ; colore rosso, che partira' da posizioni
                                    ; sempre piu' verso destra, preceduto dal
                                    ; grigio che avanzera' di conseguenza.
        dc.w        $9107,$fffe         ; wait che non cambiamo (Inizio linea)
        dc.w        $180,$111         ; colore GRIGIO (parte dall'inizio linea fino
        dc.w        $9131,$fffe         ; a questo WAIT, che noi cambiaremo...
        dc.w        $180,$200         ; dopo il quale comincia il ROSSO

;        continuiamo risparmiando spazio, osservate lo schema:

; nota: con un "dc.w $1234" mettiamo in memoria 1 word, con "dc.w $1234,$1234"
; mettiamo in memoria 2 word consecutive, ossia la longword "dc.l $12341234"
; che avremmo potuto mettere in memoria con un "dc.b $12,$34,$12,$34", dunque
; possiamo mettere in memoria anche 8 o piu' words con una sola linea dc.w!
; per esempio la linea 3 si potrebbe riscrivere con il dc.l in questo modo:
;        dc.l        $9207fffe,$1800222,$9231fffe,$1800300        ossia:
;        dc.l        $9207fffe,$01800222,$9231fffe,$01800300        con gli zeri *INIZIALI*
; fate attenzione agli zeri iniziali! un dc.w $0180 lo scrivo con dc.w $180
; semplicemente per comodita', ma lo zero esiste, va tenuto presente!
; Per chiarire, la linea 3 completa di zeri iniziali sarebbe:
;        dc.w        $9207,$fffe,$0180,$0222,$9231,$fffe,$0180,$0300 (1 word =$xxxx)
; In definitiva gli zeri iniziali "inutili" del .b, .w, .l sono FACOLTATIVI.

;            WAIT FISSI (poi grigio) - WAIT DA CAMBIARE (seguiti dal rosso)

        dc.w        $9207,$fffe,$180,$222,$9231,$fffe,$180,$300 ; linea 3
        dc.w        $9307,$fffe,$180,$333,$9331,$fffe,$180,$400 ; linea 4
        dc.w        $9407,$fffe,$180,$444,$9431,$fffe,$180,$500 ; linea 5
        dc.w        $9507,$fffe,$180,$555,$9531,$fffe,$180,$600 ; ....
        dc.w        $9607,$fffe,$180,$666,$9631,$fffe,$180,$700
        dc.w        $9707,$fffe,$180,$777,$9731,$fffe,$180,$800
        dc.w        $9807,$fffe,$180,$888,$9831,$fffe,$180,$900
        dc.w        $9907,$fffe,$180,$999,$9931,$fffe,$180,$a00
        dc.w        $9a07,$fffe,$180,$aaa,$9a31,$fffe,$180,$b00
        dc.w        $9b07,$fffe,$180,$bbb,$9b31,$fffe,$180,$c00
        dc.w        $9c07,$fffe,$180,$ccc,$9c31,$fffe,$180,$d00
        dc.w        $9d07,$fffe,$180,$ddd,$9d31,$fffe,$180,$e00
        dc.w        $9e07,$fffe,$180,$eee,$9e31,$fffe,$180,$f00
        dc.w        $9f07,$fffe,$180,$fff,$9f31,$fffe,$180,$e00
        dc.w        $a007,$fffe,$180,$eee,$a031,$fffe,$180,$d00
        dc.w        $a107,$fffe,$180,$ddd,$a131,$fffe,$180,$c00
        dc.w        $a207,$fffe,$180,$ccc,$a231,$fffe,$180,$b00
        dc.w        $a307,$fffe,$180,$bbb,$a331,$fffe,$180,$a00
        dc.w        $a407,$fffe,$180,$aaa,$a431,$fffe,$180,$900
        dc.w        $a507,$fffe,$180,$999,$a531,$fffe,$180,$800
        dc.w        $a607,$fffe,$180,$888,$a631,$fffe,$180,$700
```

```
        dc.w        $a707,$fffe,$180,$777,$a731,$fffe,$180,$600
        dc.w        $a807,$fffe,$180,$666,$a831,$fffe,$180,$500
        dc.w        $a907,$fffe,$180,$555,$a931,$fffe,$180,$400
        dc.w        $aa07,$fffe,$180,$444,$aa31,$fffe,$180,$300
        dc.w        $ab07,$fffe,$180,$333,$ab31,$fffe,$180,$200
        dc.w        $ac07,$fffe,$180,$222,$ac31,$fffe,$180,$100
        dc.w        $ad07,$fffe,$180,$111,$ad31,$fffe,$180,$000
        dc.w        $ae07,$fffe,$180,$000

;            WAIT FISSI (poi grigio) - WAIT DA CAMBIARE (seguiti dal rosso)
;
;        Come notate per ogni linea servono 2 wait, uno per aspettare l'inizio
;        della linea e uno, quello che modifichiamo, per definire in che
;        punto della linea cambiare colore, ossia passare dal grigio che e'
;        presente dalla posizione 07, al rosso che parte dopo la posizione
;        assunta dal wait che cambiamo.
;
        dc.w        $fd07,$FFFE        ; aspetto la linea $FD
        dc.w        $180,$00a        ; blu intensita' 10
        dc.w        $fe07,$FFFE        ; linea seguente
        dc.w        $180,$00f        ; blu intensita' massima (15)
        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST


        end
```

Ultima cosuccia: se non avete ancora chiaro il discorso degli zeri iniziali
affrontato prima eccovi alcune conversioni "giuste" e "sbagliate":

```
        dc.b        1,2        =        dc.w        $0102        ossia        dc.w        $102

        dc.b        42,$2        =        dc.w        $2a02        (42 decimale = $2a Hex)

        dc.b        12,$2,$12,41 = dc.w $c02,$1229 = dc.l $c021229

        dc.b        12,$22,0 = dc.w $000c,$2200 = dc.w $c,$2200 = dc.l $c2200

        dc.w        1,2,3,432 = dc.l $00010002,$000301b0 = dc.l $10002,$301b0

        dc.l        $1234567=        dc.b        1,$23,$45,$67

        dc.l        $2342        =        dc.b        0,0,$23,$42

        dc.l        4        =        dc.b        0,0,0,4
```

Attenzione all'ultimo esempio:

un dc.l 4 in memoria diventa $00000004, un dc.b 4 diventa $04
per cui mentre lo 04 nel dc.l si trova preceduto da 3 bytes $00,
nel caso del dc.b 4 il 4 si posiziona al primo posto, il che e'
completamente diverso in ASSEMBLER, nonostante si parli sempre
di un 4!!!!

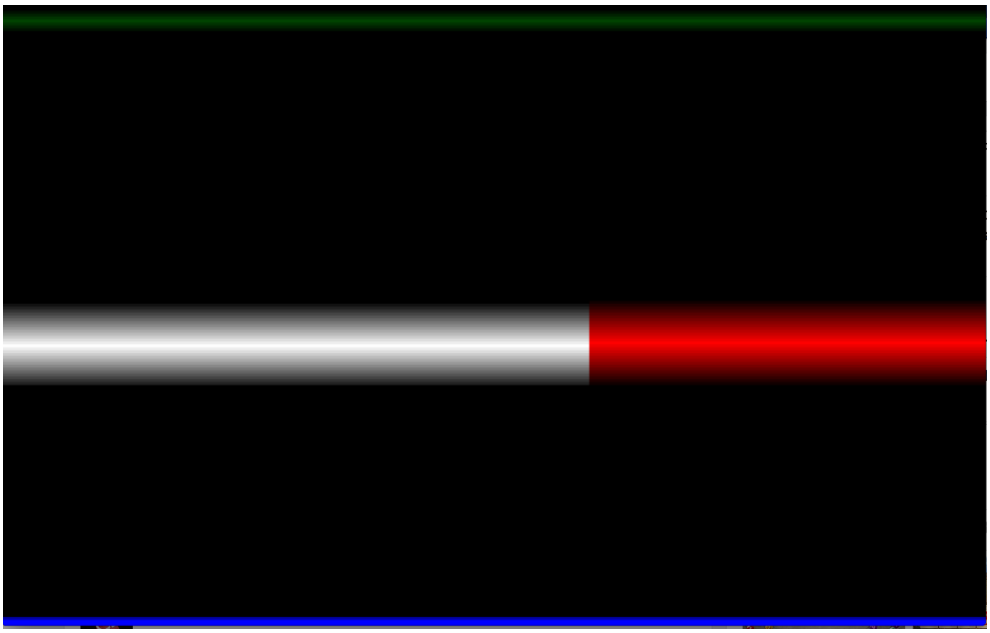


Figura 19.8: Lezione 3h

CAPITOLO 20

LEZIONE 4

20.1 Lezione4a

```
; Lezione4a.s        ROUTINE UNIVERSALE DI PUNTAMENTO BITPLANES

        SECTION        CiriBiri,CODE

Inizio:
        MOVE.L        #PIC,d0               ; in d0 mettiamo l'indirizzo della PIC,
                                ; ossia dove inizia il primo bitplane

        LEA        BPLPOINTERS,A1        ; in a1 mettiamo l'indirizzo dei
                                ; puntatori ai planes della COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
                                ; per eseguire il ciclo col DBRA
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
                                ; nella word giusta nella copperlist
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
                                ; mettendo la word ALTA al posto di quella
                                ; BASSA, permettendone la copia col move.w!!
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
                                ; nella word giusta nella copperlist
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
                                ; rimettendo a posto l'indirizzo.
        ADD.L        #40*256,d0        ; Aggiungiamo 10240 ad D0, facendolo puntare
                                ; al secondo bitplane (si trova dopo il primo)
                                ; (cioe' aggiungiamo la lunghezza di un plane)
                                ; Nei cicli seguenti al primo faremo puntare
                                ; al terzo, al quarto bitplane eccetera.

        addq.w        #8,a1                ; a1 ora contiene l'indirizzo dei prossimi
                                ; bplpointers nella copperlist da scrivere.
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

        rts        ; USCITA!!
```

```
COPPERLIST:
;        ....        ; qua metteremo i registri necessari...

;        Facciamo puntare i bitplanes direttamente mettendo nella copperlist
;        i registri $dff0e0 e seguenti qua di seguito con gli indirizzi
;        dei bitplanes che saranno messi dalla routine POINTBP

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT
;        ....
        dc.w        $FFFF,$FFFE        ; fine della copperlist

;        Ricordatevi di selezionare la directory dove si trova la figura
;        in questo caso basta scrivere: "V df0:SORGENTI2"

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        end
```

Provate a fare un "AD", ossia un DEBUG di questa routine. Debuggando fate particolare attenzione al valore di D0, visibile in alto a destra, nel momento dei 2 swap. Per verificare il funzionamento, al termine dell'esecuzione provate a controllare con un "M BPLPOINTERS" se le words sono state cambiate con l'indirizzo di PIC: SWAPPATO nelle words. (Con un "M PIC" si puo' vedere a che indirizzo e' stata caricata tramite INCBIN la PIC, che come previsto e' lunga 30720 bytes: 40*256*3).

20.2 Lezione4b

```
; Lezione4b.s        VISUALIZZAZIONE DI UNA FIGURA IN 320*256 a 3 plane (8 colori)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;*****************************************************************************
;        FACCIAMO PUNTARE I BPLPOINTERS NELLA COPPELIST AI NOSTRI BITPLANES
;*****************************************************************************


        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
                                ; ossia dove inizia il primo bitplane

        LEA        BPLPOINTERS,A1        ; in a1 mettiamo l'indirizzo dei
                                ; puntatori ai planes della COPPERLIST
```

```
        MOVEQ        #2,D1               ; numero di bitplanes -1 (qua sono 3)
                                 ; per eseguire il ciclo col DBRA
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
                                 ; nella word giusta nella copperlist
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
                                 ; mettendo la word ALTA al posto di quella
                                 ; BASSA, permettendone la copia col move.w!!
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
                                 ; nella word giusta nella copperlist
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
                                 ; rimettendo a posto l'indirizzo.
        ADD.L        #40*256,d0        ; Aggiungiamo 10240 ad D0, facendolo puntare
                                 ; al secondo bitplane (si trova dopo il primo)
                                 ; (cioe' aggiungiamo la lunghezza di un plane)
                                 ; Nei cicli seguenti al primo faremo puntare
                                 ; al terzo, al quarto bitplane eccetera.

        addq.w        #8,a1               ; a1 ora contiene l'indirizzo dei prossimi
                                 ; bplpointers nella copperlist da scrivere.
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; FMODE - Disattiva l'AGA
        move.w        #$c00,$dff106                ; BPLCON3 - Disattiva l'AGA

mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        ; Facciamo puntare gli sprite a ZERO, per eliminarli, o ce li troviamo
        ; in giro impazziti a disturbare!!!

        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000
```

```
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

; il BPLCON0 ($dff100) Per uno schermo a 3 bitplanes: (8 colori)

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)

;        Facciamo puntare i bitplanes direttamente mettendo nella copperlist
;        i registri $dff0e0 e seguenti qua di seguito con gli indirizzi
;        dei bitplanes che saranno messi dalla routine POINTBP

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT

;        Gli 8 colori della figura sono definiti qui:

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

;        Inserite qua eventuali effetti coi WAIT

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


;        Ricordatevi di selezionare la directory dove si trova la figura
;        in questo caso basta scrivere: "V df0:SORGENTI2"


PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                               ; convertita col KEFCON, fatta di
                                               ; 3 bitplanes consecutivi

        end
```

Come avrete visto non ci sono routine sincronizzate in questo esempio, ma solo le routine che puntano i bitplane e la copperlist.
Innanzitutto provate a eliminare con dei ; i puntatori degli sprite:

```
;        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000
;        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
;        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
;        dc.w        $13e,$0000
```

Noterete che ogni tanto passano come delle STRISCIATE, quelli sono sprite
senza controllo all'impazzata. Impareremo a domarli piu' avanti.

Provate ora ad aggiungere prima della fine della copperlist qualche WAIT,
e noterete come siano utili i WAIT+COLOR per AGGIUNGERE SFUMATURE ORIZZONTALI
o CAMBIARE COLORI totalmente GRATIS, ossia, con una figura a 8 colori come
questa possiamo lavorare con MOVE+WAIT facendogli uno sfondo con un centinaio
di colori sfumandoli, oppure cambiando anche i colori "in sovraimpressione",
ossia il $182, $184, $186, $188, $18a, $18c, $18e.

Come primo 'abbellimento' copiate e inserite questo pezzo prefabbricato di
sfumatura tra i colori e la fine della copperlist: (dc.w $FFFF,$FFFE)
RICORDO CHE BISOGNA SELEZIONARE IL BLOCCO CON Amiga+b, Amiga+c, poi
posizionare il cursore dove si vuole copiare il testo, e inserirlo con Amiga+i.

```
        dc.w        $a907,$FFFE         ; Aspetto la linea $a9
        dc.w        $180,$001         ; blu scurissimo
        dc.w        $aa07,$FFFE         ; linea $aa
        dc.w        $180,$002         ; blu un po' piu' intenso
        dc.w        $ab07,$FFFE         ; linea $ab
        dc.w        $180,$003         ; blu piu' chiaro
        dc.w        $ac07,$FFFE         ; prossima linea
        dc.w        $180,$004         ; blu piu' chiaro
        dc.w        $ad07,$FFFE         ; prossima linea
        dc.w        $180,$005         ; blu piu' chiaro
        dc.w        $ae07,$FFFE         ; prossima linea
        dc.w        $180,$006         ; blu a 6
        dc.w        $b007,$FFFE         ; salto 2 linee
        dc.w        $180,$007         ; blu a 7
        dc.w        $b207,$FFFE         ; sato 2 linee
        dc.w        $180,$008         ; blu a 8
        dc.w        $b507,$FFFE         ; salto 3 linee
        dc.w        $180,$009         ; blu a 9
        dc.w        $b807,$FFFE         ; salto 3 linee
        dc.w        $180,$00a         ; blu a 10
        dc.w        $bb07,$FFFE         ; salto 3 linee
        dc.w        $180,$00b         ; blu a 11
        dc.w        $be07,$FFFE         ; salto 3 linee
        dc.w        $180,$00c         ; blu a 12
        dc.w        $c207,$FFFE         ; salto 4 linee
        dc.w        $180,$00d         ; blu a 13
        dc.w        $c707,$FFFE         ; salto 7 linee
        dc.w        $180,$00e         ; blu a 14
        dc.w        $ce07,$FFFE         ; salto 6 linee
        dc.w        $180,$00f         ; blu a 15
        dc.w        $d807,$FFFE         ; salto 10 linee
        dc.w        $180,$11F         ; schiarisco...
        dc.w        $e807,$FFFE         ; salto 16 linee
        dc.w        $180,$22F         ; schiarisco...
        dc.w        $ffdf,$FFFE         ; FINE ZONA NTSC (linea $FF)
        dc.w        $180,$33F         ; schiarisco...
        dc.w        $2007,$FFFE         ; linea $20+$FF = linea $1ff (287)
        dc.w        $180,$44F         ; schiarisco...
```

Abbiamo creato dal nulla, senza effetti controproducenti, una sfumatura
portando i colori effettivi sullo schermo da 8 a 27!!!!
Aggiungiamo altri 7 colori, questa volta cambiando non il colore di sfondo,
il $dff180, ma gli altri 7 colori: inserite questo pezzo di copperlist tra
i puntatori dei bitplane e i colori: (lasciate pure l'altra modifica)

```
        dc.w        $0180,$000        ; color0
        dc.w        $0182,$550        ; color1        ; ridefiniamo il colore della
        dc.w        $0184,$ff0        ; color2        ; scritta COMMODORE! GIALLA!
        dc.w        $0186,$cc0        ; color3
        dc.w        $0188,$990        ; color4
        dc.w        $018a,$220        ; color5
        dc.w        $018c,$770        ; color6
        dc.w        $018e,$440        ; color7

        dc.w        $7007,$fffe        ; Aspettiamo la fine della scritta COMMODORE
```

Con 45 "dc.w" aggiunti alla copperlist abbiamo trasformato un'innoqua PIC di soli 8 colori in una PIC a 34 colori, superando anche il limite dei 32 colori delle pic a 5 bitplanes!!!

Solo programmando le copperlist in assembler si puo' sfruttare al massimo la grafica di Amiga: ora potreste anche fare delle figure a 320 colori puliti puliti semplicemente cambiando l'intera palette di una figura a 32 colori 10 volte, mettendo un wait+palette ogni 25 linee...
Ora forse vi spiegherete come mai certi giochi hanno 64, 128 o piu' colori sullo schermo!!! Hanno delle copperlist lunghissime dove cambiano colore a diverse altezze del video!

Fatevi un po' di modifiche, che fanno sempre bene, e se vi va provate a mettere in "sottofondo" gli esempi con le barrette della Lezione3, basta caricarseli in altri buffer e inserire i pezzi di routine e di copperlist giusti, e' un buon allenamento. Provate a far camminare la barretta "sotto" il disegno, se ci riuscite siete tosti.

20.3 Lezione4c

```
; Lezione4c.s        FUSIONE DI 3 EFFETTI COPPER + FIGURA AD 8 COLORI

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;*****************************************************************************
;        FACCIAMO PUNTARE I BPLPOINTERS NELLA COPPELIST AI NOSTRI BITPLANES
;*****************************************************************************


        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
                                ; ossia dove inizia il primo bitplane

        LEA        BPLPOINTERS,A1        ; in a1 mettiamo l'indirizzo dei
                                ; puntatori ai planes della COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
                                ; per eseguire il ciclo col DBRA
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
                                ; nella word giusta nella copperlist
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
```

```
                            ; mettendo la word ALTA al posto di quella
                            ; BASSA, permettendone la copia col move.w!!
        move.w      d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
                            ; nella word giusta nella copperlist
        swap        d0              ; scambia le 2 word di d0 (es: 3412 > 1234)
                            ; rimettendo a posto l'indirizzo.
        ADD.L       #40*256,d0       ; Aggiungiamo 10240 ad D0, facendolo puntare
                            ; al secondo bitplane (si trova dopo il primo)
                            ; (cioe' aggiungiamo la lunghezza di un plane)
                            ; Nei cicli seguenti al primo faremo puntare
                            ; al terzo, al quarto bitplane eccetera.

        addq.w      #8,a1               ; a1 ora contiene l'indirizzo dei prossimi
                            ; bplpointers nella copperlist da scrivere.
        dbra        d1,POINTBP      ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;

        move.l      #COPPERLIST,$dff080       ; Puntiamo la nostra COP
        move.w      d0,$dff088              ; Facciamo partire la COP

        move.w      #0,$dff1fc              ; FMODE - Disattiva l'AGA
        move.w      #$c00,$dff106              ; BPLCON3 - Disattiva l'AGA

mouse:
        cmpi.b      #$ff,$dff006        ; Siamo alla linea 255?
        bne.s       mouse               ; Se non ancora, non andare avanti

        bsr.w       muovicopper        ; barra rossa sotto linea $ff
        bsr.w       CopperDestSin       ; Routine di scorrimento destra/sinistra
        BSR.w       scrollcolors       ; scorrimento ciclico dei colori

Aspetta:
        cmpi.b      #$ff,$dff006        ; Siamo alla linea 255?
        beq.s       Aspetta               ; Se si, non andare avanti, aspetta la linea
                            ; seguente, altrimenti MuoviCopper viene
                            ; rieseguito

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s       mouse               ; se no, torna a mouse:


        move.l      OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w      d0,$dff088              ; facciamo partire la vecchia cop

        move.l      4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l      gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA


;       Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                       ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0
```

```
; ***************************************************************************
; *                BARRA A SCORRIMENTO ORIZZONTALE (Lezione3h.s)                *
; ***************************************************************************

CopperDESTSIN:
        CMPI.W        #85,DestraFlag                ; VAIDESTRA eseguita 85 volte?
        BNE.S        VAIDESTRA                ; se non ancora, rieseguila
        CMPI.W        #85,SinistraFlag        ; VAISINISTRA eseguita 85 volte?
        BNE.S        VAISINISTRA                ; se non ancora, rieseguila
        CLR.W        DestraFlag        ; la routine VAISINISTRA e' stata eseguita
        CLR.W        SinistraFlag        ; 85 volte, riparti
        RTS                        ; TORNIAMO AL LOOP mouse


VAIDESTRA:                        ; questa routine sposta la barra verso DESTRA
        lea        CopBar+1,A0        ; Mettiamo in A0 l'indirizzo del primo XX
        move.w        #29-1,D2        ; dobbiamo cambiare 29 wait (usiamo un DBRA)
DestraLoop:
        addq.b        #2,(a0)                ; aggiungiamo 2 alla coordinata X del wait
        ADD.W        #16,a0                ; andiamo al prossimo wait da cambiare
        dbra        D2,DestraLoop        ; ciclo eseguito d2 volte
        addq.w        #1,DestraFlag        ; segnamo che abbiamo eseguito VAIDESTRA
        RTS                        ; TORNIAMO AL LOOP mouse


VAISINISTRA:                        ; questa routine sposta la barra verso SINISTRA
        lea        CopBar+1,A0
        move.w        #29-1,D2        ; dobbiamo cambiare 29 wait
SinistraLoop:
        subq.b        #2,(a0)                ; sottraiamo 2 alla coordinata X del wait
        ADD.W        #16,a0                ; andiamo al prossimo wait da cambiare
        dbra        D2,SinistraLoop        ; ciclo eseguito d2 volte
        addq.w        #1,SinistraFlag ; Annotiamo lo spostamento
        RTS                        ; TORNIAMO AL LOOP mouse


DestraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w        0        ; che e' stata eseguita VAIDESTRA

SinistraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w    0        ; che e' stata eseguita VAISINISTRA

; ***************************************************************************
; *                BARRA ROSSA SOTTO LA LINEA $FF (Lezione3f.s)                *
; ***************************************************************************

MuoviCopper:
        LEA        BARRA,a0
        TST.B        SuGiu                ; Dobbiamo salire o scendere?
        beq.w        VAIGIU
        cmpi.b        #$0a,(a0)        ; siamo arrivati alla linea $0a+$ff? (265)
        beq.s        MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        subq.b        #1,(a0)
        subq.b        #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        subq.b        #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        subq.b        #1,8*3(a0)
        subq.b        #1,8*4(a0)
        subq.b        #1,8*5(a0)
        subq.b        #1,8*6(a0)
        subq.b        #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        subq.b        #1,8*8(a0)        ; barra rossa ogni volta per farla salire!
```

```
        subq.b        #1,8*9(a0)
        rts

MettiGiu:
        clr.b         SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                          ; fara' saltare alla routine VAIGIU, e
                                   ; la barra scedera'

VAIGIU:
        cmpi.b        #$2c,8*9(a0)         ; siamo arrivati alla linea $2c?
        beq.s         MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        addq.b        #1,(a0)
        addq.b        #1,8(a0)         ; ora cambiamo gli altri wait: la distanza
        addq.b        #1,8*2(a0)         ; tra un wait e l'altro e' di 8 bytes
        addq.b        #1,8*3(a0)
        addq.b        #1,8*4(a0)
        addq.b        #1,8*5(a0)
        addq.b        #1,8*6(a0)
        addq.b        #1,8*7(a0)         ; qua dobbiamo modificare tutti i 9 wait della
        addq.b        #1,8*8(a0)         ; barra rossa ogni volta per farla scendere!
        addq.b        #1,8*9(a0)
        rts

MettiSu:
        move.b        #$ff,SuGiu         ; Quando la label SuGiu non e' a zero,
        rts                          ; significa che dobbiamo risalire.


SuGiu:
        dc.b         0,0

; ***************************************************************************
; *                SCORRIMENTO CICLICO DEI COLORI (Lezione3E.s)                  *
; ***************************************************************************

Scrollcolors:
        move.w        col2,col1         ; col2 copiato in col1
        move.w        col3,col2         ; col3 copiato in col2
        move.w        col4,col3         ; col4 copiato in col3
        move.w        col5,col4         ; col5 copiato in col4
        move.w        col6,col5         ; col6 copiato in col5
        move.w        col7,col6         ; col7 copiato in col6
        move.w        col8,col7         ; col8 copiato in col7
        move.w        col9,col8         ; col9 copiato in col8
        move.w        col10,col9         ; col10 copiato in col9
        move.w        col11,col10         ; col11 copiato in col10
        move.w        col12,col11         ; col12 copiato in col11
        move.w        col13,col12         ; col13 copiato in col12
        move.w        col14,col13         ; col14 copiato in col13
        move.w        col1,col14         ; col1 copiato in col14
        rts

; ***************************************************************************
; *                                SUPER COPPERLIST                           *
; ***************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        ; Facciamo puntare gli sprite a ZERO, per eliminarli, o ce li troviamo
        ; in giro impazziti a disturbare!!!
```

```
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

; Il BPLCON0 per uno schermo a 3 bitplanes: (8 colori)

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)

;        Facciamo puntare i bitplanes direttamente mettendo nella copperlist
;        i registri $dff0e0 e seguenti qua di seguito con gli indirizzi
;        dei bitplanes che saranno messi dalla routine POINTBP

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT

;        L'effetto di Lezione3e.s spostato piu' in ALTO

        dc.w        $3a07,$fffe        ; aspettiamo la linea 154 ($9a in esadecimale)
        dc.w        $180                ; REGISTRO COLOR0
col1:
        dc.w        $0f0                ; VALORE DEL COLOR 0 (che sara' modificato)
        dc.w        $3b07,$fffe ; aspettiamo la linea 155 (non sara' modificata)
        dc.w        $180                ; REGISTRO COLOR0 (non sara' modificato)
col2:
        dc.w        $0d0                ; VALORE DEL COLOR 0 (sara' modificato)
        dc.w        $3c07,$fffe        ; aspettiamo la linea 156 (non modificato,ecc.)
        dc.w        $180                ; REGISTRO COLOR0
col3:
        dc.w        $0b0                ; VALORE DEL COLOR 0
        dc.w         $3d07,$fffe        ; aspettiamo la linea 157
        dc.w        $180                ; REGISTRO COLOR0
col4:
        dc.w        $090                ; VALORE DEL COLOR 0
        dc.w        $3e07,$fffe        ; aspettiamo la linea 158
        dc.w        $180                ; REGISTRO COLOR0
col5:
        dc.w        $070                ; VALORE DEL COLOR 0
        dc.w        $3f07,$fffe        ; aspettiamo la linea 159
        dc.w        $180                ; REGISTRO COLOR0
col6:
        dc.w        $050                ; VALORE DEL COLOR 0
        dc.w        $4007,$fffe        ; aspettiamo la linea 160
        dc.w        $180                ; REGISTRO COLOR0
col7:
        dc.w        $030                ; VALORE DEL COLOR 0
        dc.w        $4107,$fffe        ; aspettiamo la linea 161
        dc.w        $180                ; color0... (ora avete capito i commenti,
col8:                                ; posso anche smettere di metterli da qua!)
```

```
        dc.w        $030
        dc.w        $4207,$fffe        ; linea 162
        dc.w        $180
col9:
        dc.w        $050
        dc.w        $4307,$fffe        ;  linea 163
        dc.w        $180
col10:
        dc.w        $070
        dc.w        $4407,$fffe        ;  linea 164
        dc.w        $180
col11:
        dc.w        $090
        dc.w        $4507,$fffe        ;  linea 165
        dc.w        $180
col12:
        dc.w        $0b0
        dc.w        $4607,$fffe        ;  linea 166
        dc.w        $180
col13:
        dc.w        $0d0
        dc.w        $4707,$fffe        ;  linea 167
        dc.w        $180
col14:
        dc.w        $0f0
        dc.w         $4807,$fffe        ;  linea 168

        dc.w         $180,$0000        ; Decidiamo il colore NERO per la parte
                                ; di schermo sotto l'effetto


        dc.w        $0180,$000        ; color0
        dc.w        $0182,$550        ; color1        ; ridefiniamo il colore della
        dc.w        $0184,$ff0        ; color2        ; scritta COMMODORE! GIALLA!
        dc.w        $0186,$cc0        ; color3
        dc.w        $0188,$990        ; color4
        dc.w        $018a,$220        ; color5
        dc.w        $018c,$770        ; color6
        dc.w        $018e,$440        ; color7

        dc.w        $7007,$fffe        ; Aspettiamo la fine della scritta COMMODORE

;        Gli 8 colori della figura sono definiti qui:

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

;        EFFETTO DELLA LEZIONE3h.s

        dc.w        $9007,$fffe        ; aspettiamo l'inizio della linea
        dc.w        $180,$000        ; grigio al minimo, ossia NERO!!!
CopBar:
        dc.w        $9031,$fffe        ; wait che cambiamo ($9033,$9035,$9037...)
        dc.w        $180,$100        ; colore rosso
        dc.w        $9107,$fffe        ; wait che non cambiamo (Inizio linea)
        dc.w        $180,$111        ; colore GRIGIO (parte dall'inizio linea fino
```

```
        dc.w        $9131,$fffe        ; a questo WAIT, che noi cambiaremo...
        dc.w        $180,$200        ; dopo il quale comincia il ROSSO

;            WAIT FISSI (poi grigio) - WAIT DA CAMBIARE (seguiti dal rosso)

        dc.w        $9207,$fffe,$180,$222,$9231,$fffe,$180,$300 ; linea 3
        dc.w        $9307,$fffe,$180,$333,$9331,$fffe,$180,$400 ; linea 4
        dc.w        $9407,$fffe,$180,$444,$9431,$fffe,$180,$500 ; linea 5
        dc.w        $9507,$fffe,$180,$555,$9531,$fffe,$180,$600 ; ....
        dc.w        $9607,$fffe,$180,$666,$9631,$fffe,$180,$700
        dc.w        $9707,$fffe,$180,$777,$9731,$fffe,$180,$800
        dc.w        $9807,$fffe,$180,$888,$9831,$fffe,$180,$900
        dc.w        $9907,$fffe,$180,$999,$9931,$fffe,$180,$a00
        dc.w        $9a07,$fffe,$180,$aaa,$9a31,$fffe,$180,$b00
        dc.w        $9b07,$fffe,$180,$bbb,$9b31,$fffe,$180,$c00
        dc.w        $9c07,$fffe,$180,$ccc,$9c31,$fffe,$180,$d00
        dc.w        $9d07,$fffe,$180,$ddd,$9d31,$fffe,$180,$e00
        dc.w        $9e07,$fffe,$180,$eee,$9e31,$fffe,$180,$f00
        dc.w        $9f07,$fffe,$180,$fff,$9f31,$fffe,$180,$e00
        dc.w        $a007,$fffe,$180,$eee,$a031,$fffe,$180,$d00
        dc.w        $a107,$fffe,$180,$ddd,$a131,$fffe,$180,$c00
        dc.w        $a207,$fffe,$180,$ccc,$a231,$fffe,$180,$b00
        dc.w        $a307,$fffe,$180,$bbb,$a331,$fffe,$180,$a00
        dc.w        $a407,$fffe,$180,$aaa,$a431,$fffe,$180,$900
        dc.w        $a507,$fffe,$180,$999,$a531,$fffe,$180,$800
        dc.w        $a607,$fffe,$180,$888,$a631,$fffe,$180,$700
        dc.w        $a707,$fffe,$180,$777,$a731,$fffe,$180,$600
        dc.w        $a807,$fffe,$180,$666,$a831,$fffe,$180,$500
        dc.w        $a907,$fffe,$180,$555,$a931,$fffe,$180,$400
        dc.w        $aa07,$fffe,$180,$444,$aa31,$fffe,$180,$301
        dc.w        $ab07,$fffe,$180,$333,$ab31,$fffe,$180,$202
        dc.w        $ac07,$fffe,$180,$222,$ac31,$fffe,$180,$103
        dc.w        $ad07,$fffe,$180,$113,$ad31,$fffe,$180,$004

        dc.w        $ae07,$FFFE        ; prossima linea
        dc.w        $180,$006        ; blu a 6
        dc.w        $b007,$FFFE        ; salto 2 linee
        dc.w        $180,$007        ; blu a 7
        dc.w        $b207,$FFFE        ; sato 2 linee
        dc.w        $180,$008        ; blu a 8
        dc.w        $b507,$FFFE        ; salto 3 linee
        dc.w        $180,$009        ; blu a 9
        dc.w        $b807,$FFFE        ; salto 3 linee
        dc.w        $180,$00a        ; blu a 10
        dc.w        $bb07,$FFFE        ; salto 3 linee
        dc.w        $180,$00b        ; blu a 11
        dc.w        $be07,$FFFE        ; salto 3 linee
        dc.w        $180,$00c        ; blu a 12
        dc.w        $c207,$FFFE        ; salto 4 linee
        dc.w        $180,$00d        ; blu a 13
        dc.w        $c707,$FFFE        ; salto 7 linee
        dc.w        $180,$00e        ; blu a 14
        dc.w        $ce07,$FFFE        ; salto 6 linee
        dc.w        $180,$00f        ; blu a 15
        dc.w        $d807,$FFFE        ; salto 10 linee
        dc.w        $180,$11F        ; schiarisco...
        dc.w        $e807,$FFFE        ; salto 16 linee
        dc.w        $180,$22F        ; schiarisco...

;        Effetto della lezione3f.s

        dc.w        $ffdf,$fffe        ; ATTENZIONE! WAIT ALLA FINE LINEA $FF!
```

```
                                ; i wait dopo questo sono sotto la linea
                                ; $FF e ripartono da $00!!

        dc.w        $0107,$FFFE        ; una barretta fissa verde SOTTO la linea $FF!
        dc.w        $180,$010
        dc.w        $0207,$FFFE
        dc.w        $180,$020
        dc.w        $0307,$FFFE
        dc.w        $180,$030
        dc.w        $0407,$FFFE
        dc.w        $180,$040
        dc.w        $0507,$FFFE
        dc.w        $180,$030
        dc.w        $0607,$FFFE
        dc.w        $180,$020
        dc.w        $0707,$FFFE
        dc.w        $180,$010
        dc.w        $0807,$FFFE
        dc.w        $180,$000

BARRA:
        dc.w        $0907,$FFFE        ; aspetto la linea $79
        dc.w        $180,$300        ; inizio la barra rossa: rosso a 3
        dc.w        $0a07,$FFFE        ; linea seguente
        dc.w        $180,$600        ; rosso a 6
        dc.w        $0b07,$FFFE
        dc.w        $180,$900        ; rosso a 9
        dc.w        $0c07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $0d07,$FFFE
        dc.w        $180,$f00        ; rosso a 15 (al massimo)
        dc.w        $0e07,$FFFE
        dc.w        $180,$c00        ; rosso a 12
        dc.w        $0f07,$FFFE
        dc.w        $180,$900        ; rosso a 9
        dc.w        $1007,$FFFE
        dc.w        $180,$600        ; rosso a 6
        dc.w        $1107,$FFFE
        dc.w        $180,$300        ; rosso a 3
        dc.w        $1207,$FFFE
        dc.w        $180,$000        ; colore NERO

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST


; ***************************************************************************
; *                        FIGURA AD 8 COLORI 320x256                       *
; ***************************************************************************

;        Ricordatevi di selezionare la directory dove si trova la figura
;        in questo caso basta scrivere: "V df0:SORGENTI2"


PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        end
```

In questo esempio non c'e' nulla di nuovo, ma abbiamo messo insieme molti
degli effetti copper fin qui studiati: Lezione3h.s, Lezione3f.s, Lezione3e.s,

semplicemente caricando quei sorgenti in altri buffer di testo, copiandone la routine e la parte di copperlist dell'effetto: le routines come si puo' notare sono una sotto l'altra nell'ordine con cui ho caricato gli esempi, mentre i wait delle copperlist vanno "AGGIUNTATI" secondo un preciso ordine, in modo che non si sovrappongano: infatti ho dovuto spostare piu' in altro i wait dell'effetto Lezione3f.s, mentre gli altri 2 li ho potuti lasciare uguali. Bastera' poi che nel loop sincronizzato si richiamino le routines:

```
        bsr.w       muovicopper        ; barra rossa sotto linea $ff
        bsr.w       CopperDestSin        ; Routine di scorrimento destra/sinistra
        BSR.w       scrollcolors        ; scorrimento ciclico dei colori
```

Spesso si programmano le singole routines separatamente per poi metterle insieme come in questo esempio; e' bene esercitarsi a montare e smontare demo grafiche come in questo esempio, perche' in fondo buona parte della programmazione e' costitutita dal montaggio delle routines. Ogni routine poi puo' essere riutilizzata in molti listati, con semplici modifiche: per esempio il programmatore dei TEAM 17 sicuramente ha usato le stesse routines di gestione joystick e di caricamento da disco su tutti i suoi giochi, e probabilmente le routines che spostano i personaggi sullo schermo sono derivate l'una dall'altra con poche modifiche. Ogni routine che programmate o che trovate in giro puo' servirvi molte volte, sia come esempio sia per metterla proprio in vostri programmi. Se aveste tutte le routines necessarie alla programmazione di un gioco separate, ipotizziamo un joystick.s, un caricadisco.s, un suonamusica.s, un scrollaschermo.s, eccetera, fare il gioco si limiterebbe ad una operazione simile a chi apparecchia la tavola: cioe' mettere i tovaglioli, i piatti, le posate al punto giusto, cosi' dovreste mettere insieme il gioco come un puzzle, cosa che richiederebbe comunque la conoscenza almeno del funzionamento delle routines.
Il problema di tante demo e di tanti giochi infatti sta nel fatto che le routines sono ben combinate, la grafica e il suono se le sono fatte, ma viene il sospetto che le routines provengano da altri programmatori, rubate o concesse. D'altronde se il gioco funziona, cosa importa? Sara' sempre un bel gioco ma simile a qualche altro, un incrocio. Quando le routines uno se le programma da solo, si riconosce sempre perche' o le ha fatte peggio degli altri, o le ha fatte meglio. Dunque i giochi brutti e quelli bellissimi sono i piu' "ONESTAMENTE" programmati. Ma vi consiglio di lasciare da parte l'orgoglio per ora che state imparando, non credo che possiate innovare la programmazione Amiga ora! Dunque scomponete e riaggiuntate le routines che trovate nel corso come in questo listato, lo scopo e' imparare, e non c'e' modo migliore di imparare che aggiuntare e smontare routines. Basta che poi non andiate in giro con le MIE routines a dire che le avete programmate voi del tutto. Quando avrete finito questo corso, allora sarete in grado di farvele, e magari di avere idee innovative, l'assembler non pone limiti.

CAPITOLO 21

LEZIONE 5

21.1 Lezione5a

```
; Lezione5a.s        SCORRIMENTO DI UNA FIGURA A DESTRA E SINISTRA COL $dff102

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
```

```
        move.w        #$c00,$dff106               ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        Aspetta                ; la routine dello scroll, bloccandolo

        bsr.s        MuoviCopper        ; fa scorrere col $dff102 la figura a destra
                            ; e a sinistra (massimo 16 pixel)

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Questa routine e' simile a quella di Lezione3d.s, in questo caso
;        modifichiamo il valore del registro di scroll BPLCON1 $dff102 per
;        scorrere in avanti ed indietro la figura.
;        Essendo possibile agire separatamente sui bitplanes pari e su quelli
;        dispari, per spostare tutti i bitplanes dobbiamo spostarli
;        contemporaneamente: $0011,$0022,$0033 anziche' $0001,$0002,$0003 che
;        sposterebbe solo i bitplanes dispari (1,3,5), o $0010,$0020,$0030 che
;        sposterebbe solo i bitplanes pari (2,4,6).
;        Provate con un "=c 102" per vedere i bit del $dff102

MuoviCopper:
        TST.B        FLAG                ; Dobbiamo avanzare o indietreggiare? se
                            ; FLAG e' azzerata, (cioe' il TST verifica il
                            ; BEQ)
                            ; allora saltiamo a AVANTI, se invece e' a $FF
                            ; (se cioe' questo TST non e' verificato)
                            ; continuiamo indietreggiando (con dei sub)
        beq.w        AVANTI
        cmpi.b        #$00,MIOCON1        ; siamo arrivati alla posizione normale, ossia
                            ; tutto indietro?
        beq.s        MettiAvanti        ; se si, dobbiamo avanzare!
        sub.b        #$11,MIOCON1        ; sottraiamo 1 allo scroll dei bitplanes
        rts                        ; dispari ($ff,$ee,$dd,$cc,$bb,$aa,$99....)
```

```
                                ; andando a SINISTRA
MettiAvanti:
        clr.b        FLAG               ; Azzerando FLAG, al TST.B FLAG il BEQ
        rts                          ; fara' saltare alla routine AVANTI, e
                                ; la figura avanzera' (verso destra)

AVANTI:
        cmpi.b        #$ff,MIOCON1        ; siamo arrivati allo scroll massimo in
                                ; avanti, ossia $FF? ($f pari e $f dispari)
        beq.s        MettiIndietro        ; se si, siamo dobbiamo tornare indietro
        add.b        #$11,MIOCON1        ; aggiungiamo 1 allo scroll dei bitplanes
                                ; pari e dispari ($11,$22,$33,$44 etc..)
        rts                          ; ANDANDO A DESTRA

MettiIndietro:
        move.b        #$ff,FLAG        ; Quando la label FLAG non e' a zero,
        rts                          ; significa che dobbiamo indietreggiare
                                ; verso sinistra

;        Questo byte e' un FLAG, ossia serve per indicare se andare avanti o
;        indietro.

FLAG:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop


        dc.w        $102                ; BplCon1 - IL REGISTRO
        dc.b        $00                ; BplCon1 - IL BYTE NON UTILIZZATO!!!
MIOCON1:
        dc.b        $00                ; BplCon1 - IL BYTE UTILIZZATO!!!


        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                ; 5432109876543210        ; BPLCON0:
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
```

```
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        end
```

Spostare lo schermo in avanti di 16 pixel sull'Amiga e' uno scherzo! basta modificare un byte, quello del $dff102, e il gioco e' fatto. Su altri sistemi grafici di computer come il PC MSDOS invece bisogna proprio modificare tutta la figura e "spostarla", con moltissime istruzioni che rallentano tutto. Inoltre si possono spostare separatamente i planes pari e quelli dispari, in modo da creare facilmente effetti di parallasse, basta far scorrere piu' lentamente lo sfondo, fatto dai bitplanes dispari, e piu' velocemente il primo piano, fatto ad esempio dai bitplanes pari. Non per niente per fare una parallasse sul PC occorre fare routines complicatissime e lente.
Verifichiamo che e' possibile scorrere i bitplanes pari e dispari separatamente con queste due modifiche; per scorrere SOLO i bitplanes PARI (qua c'e' il 2 solamente) cambiate queste istruzioni

```
        sub.b        #$11,MIOCON1        ; sottraiamo 1 allo scroll dei bitplanes

        cmpi.b        #$ff,MIOCON1        ; siamo arrivati allo scroll massimo in

        add.b        #$11,MIOCON1        ; aggiungiamo 1 allo scroll dei bitplanes
                                ; pari e dispari ($11,$22,$33,$44 etc..)
```

in questo modo:

```
        sub.b        #$10,MIOCON1        ; solo i planes PARI!

        cmpi.b        #$f0,MIOCON1

        add.b        #$10,MIOCON1
```

Noterete che si muove un solo bitplane, il 2, mentre il primo ed il terzo rimangono al loro posto. Nello spostarsi il bitplane 2 rimane "allo scoperto", ossia perde la sovrapposizione con gli altri 2 mostrando la sua "VERA FACCIA", e assumendo il COLOR2, che e' a $FFF nella copperlist come potete vedere, infatti e' bianco. Assume il color2 perche' spostandosi il bitplane 2 si trova "solo" con lo sfondo, ossia: %010, con i bitplane 1 e 3 azzerati. Il numero binario %010 equivale a 2, dunque il suo colore sara' deciso dal registro colore 2, il $dff184. cambiate nella copperlist il suo valore e verificherete che il bitplane 2 "da solo" e' controllato proprio da quel registro:

```
        dc.w        $0184,$fff        ; color2
```

Infatti mettendo, ad esempio, un $ff0, diventera' giallo. D'altronde la figura rimane "BUCATA" nei punti dove il bitplane2 "SE NE VA", potete vederlo meglio premendo il tasto destro che blocca lo scorrimento: in particolare i buchi

si notano dove compare il BIANCO, ossia dove c'era solo il bitplane2 senza
sovrapposizioni. In altri casi anziche' formarsi un BUCO cambia il colore.

Per far scorrere solo i bitplanes DISPARI (1 e 3 nella nostra figura), invece,
modificate la routine cosi':

```
        subq.b        #$01,MIOCON1        ; solo i planes DISPARI!

        cmpi.b        #$0f,MIOCON1

        addq.b        #$01,MIOCON1
```

In questo caso rimane fermo il bitplane2, l'unico pari, e si muovono i plane
1 e 3, i dispari.
Con questo esempi avete potuto verificare anche il metodo della sovrapposizione
dei bitplane per visualizzare i vari colori.

21.2 Lezione5b

```
; Lezione5b.s        SCORRIMENTO DI UNA FIGURA A DESTRA E SINISTRA COL $dff102

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        bsr.s        MuoviCopper        ; fa scorrere col $dff102 la figura a destra
                                ; e a sinistra (massimo 16 pixel), qua
```

```
                            ; la scritta COMMODORE

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        Aspetta               ; la routine dello scroll, bloccandolo

        bsr.w        MuoviCopper2        ; fa scorrere col $dff102 la figura a destra
                            ; e a sinistra (massimo 16 pixel), qua la
                            ; scritta AMIGA

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Questa routine sposta la scritta "COMMODORE", agendo su MIOCON1

MuoviCopper:
        TST.B        FLAG                ; Dobbiamo avanzare o indietreggiare? se
                            ; FLAG e' azzerata, (cioe' il TST verifica il
                            ; BEQ)
                            ; allora saltiamo a AVANTI, se invece e' a $FF
                            ; (se cioe' questo TST non e' verificato)
                            ; continuiamo indietreggiando (con dei sub)
        beq.w        AVANTI
        cmpi.b        #$00,MIOCON1        ; siamo arrivati alla posizione normale, ossia
                            ; tutto indietro?
        beq.s        MettiAvanti        ; se si, dobbiamo avanzare!
        sub.b        #$11,MIOCON1        ; sottraiamo 1 allo scroll dei bitplanes
        rts                        ; dispari ($ff,$ee,$dd,$cc,$bb,$aa,$99....)

MettiAvanti:
        clr.b        FLAG                ; Azzerando FLAG, al TST.B FLAG il BEQ
        rts                        ; fara' saltare alla routine AVANTI, e
                            ; la figura avanzera' (verso destra)

AVANTI:
        cmpi.b        #$ff,MIOCON1        ; siamo arrivati allo scroll massimo in
                            ; avanti, ossia $FF? ($f pari e $f dispari)
        beq.s        MettiIndietro        ; se si, siamo dobbiamo tornare indietro
        add.b        #$11,MIOCON1        ; aggiungiamo 1 allo scroll dei bitplanes
```

```
                                ; pari e dispari ($11,$22,$33,$44 etc..)
        rts

MettiIndietro:
        move.b        #$ff,FLAG        ; Quando la label FLAG non e' a zero,
        rts                            ; significa che dobbiamo indietreggiare
                                ; verso sinistra

;        Questo byte e' un FLAG, ossia serve per indicare se andare avanti o
;        indietro.

FLAG:
        dc.b        0,0

;*****************************************************************************

;        Questa routine sposta la scritta "AMIGA", agendo su MIACON1

MuoviCopper2:
        TST.B        FLAG2                ; Dobbiamo avanzare o indietreggiare?
        beq.w        AVANTI2
        cmpi.b        #$00,MIACON1        ; siamo arrivati alla posizione normale?
        beq.s        MettiAvanti2        ; se si, dobbiamo avanzare!
        sub.b        #$11,MIACON1        ; sottraiamo 1 allo scroll dei bitplanes
        rts                            ; ($ff,$ee,$dd,$cc,$bb,$aa,$99....)

MettiAvanti2:
        clr.b        FLAG2                ; Azzerando FLAG, al TST.B FLAG il BEQ
        rts                            ; fara' saltare alla routine AVANTI

AVANTI2:
        cmpi.b        #$ff,MIACON1        ; siamo arrivati allo scroll massimo in
                                ; avanti, ossia $FF? ($f pari e $f dispari)
        beq.s        MettiIndietro2        ; se si, siamo dobbiamo tornare indietro
        add.b        #$11,MIACON1        ; aggiungiamo 1 allo scroll dei bitplanes
                                ; pari e dispari ($11,$22,$33,$44 etc..)
        rts

MettiIndietro2:
        move.b        #$ff,FLAG2        ; Quando la label FLAG non e' a zero,
        rts                            ; significa che dobbiamo indietreggiare.

Finito2:
        rts

;        Questo byte e' un FLAG, ossia serve per indicare se andare avanti o
;        indietro.

FLAG2:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
```

```
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop

        dc.w        $102               ; BplCon1 - IL REGISTRO
        dc.b        $00               ; BplCon1 - IL BYTE NON UTILIZZATO!!!
MIOCON1:
        dc.b        $00               ; BplCon1 - IL BYTE UTILIZZATO!!!

        dc.w        $104,0              ; BplCon2
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210        ; BPLCON0:
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                      ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $7007,$fffe        ; Aspettiamo fino sotto la scritta "COMMODORE"

        dc.w        $102               ; BplCon1 - IL REGISTRO
        dc.b        $00               ; BplCon1 - IL BYTE NON UTILIZZATO!!!
MIACON1:
        dc.b        $ff               ; BplCon1 - IL BYTE UTILIZZATO!!!


        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                      ; convertita col KEFCON, fatta di
                                      ; 3 bitplanes consecutivi

        end
```

Questo esempio e' stato ottenuto copiando la routine Muovicopper, e cambiando le sue label aggiungendoci un 2 per "cambiargli nome", per non riscriverla tutta. Spesso per aggiungere routines simili si ricorre alla copia del pezzo interessato con Amiga+b+c+i, poi si cambia il nome alle label. Per quanto riguarda la copperlist e' bastato aggiungere un'altro $dff102, il cui nome e' MIACON1, dopo un WAIT $7007, ossia sotto la scritta commodore, per cui agisce sulla parte sottostante di figura, che e' il disegno "AMIGA".
Per creare la "DISCORDANZA" di movimento, per cui una parte va a destra quando l'altra va a sinistra e viceversa, e' bastato far partire il loop da $FF anziche' da $00, ossia dalla posizione 15, per cui i due cicli Muovicopper e Muovicopper2 proseguino dalle 2 posizioni opposte.

```
        dc.w        $102               ; BplCon1 - IL REGISTRO
        dc.b        $00               ; BplCon1 - IL BYTE NON UTILIZZATO!!!
```

```
MIOCON1:
        dc.b        $00                ; BplCon1 - IL BYTE UTILIZZATO!!!

        ...

        dc.w        $102                ; BplCon1 - IL REGISTRO
        dc.b        $00                ; BplCon1 - IL BYTE NON UTILIZZATO!!!
MIACON1:
        dc.b        $ff                ; BplCon1 - IL BYTE UTILIZZATO!!!

Provate a cambiare il byte MIACON1:, anziche' $ff provate con $55 e $aa o altri
valori, e risultera' piu' chiaro.

Col tasto destro del mouse si blocca solo il secondo $102.
Provate a cambiare il Wait per far avvenire la differenza di scroll in altre
posizioni, ad esempio:


        dc.w        $a007,$fffe

Fa "dividere" la figura nel mezzo della scritta "AMIGA".
```

21.3 Lezione5c

```
; Lezione5c.s        SCORRIMENTO DI UNA FIGURA IN ALTO E IN BASSO MODIFICANDO I
;                PUNTATORI AI PITPLANES NELLA COPPERLIST

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA
```

```
mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s         mouse               ; Se non ancora, non andare avanti

        btst         #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s         Aspetta               ; la routine dello scroll, bloccandolo


        bsr.w         MuoviCopper        ; fa scorrere la figura in alto e in basso
                                  ; di una linea alla volta cambiando i
                                  ; puntatori ai bitplanes in copperlist

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s         Aspetta               ; Se si, non andare avanti, aspetta!

        btst         #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s         mouse               ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr         -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                         ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l         0        ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l         0


;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)
;        La struttura e' simile a quella di Lezione3d.s
;        Per prima cosa mettiamo l'indirizzo che stanno puntato i BPLPOINTERS
;        in d0, poi aggiungiamo o sottraiamo 40 a d0, e infine per modificare
;        in copperlist i BPLPOINTERS dobbiamo "ripuntare" il valore cambiato
;        in d0 con la stessa routine POINTBP.

MuoviCopper:
        LEA         BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap         d0                ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0        ; in d0 - il contrario della routine che
                                  ; punta i bitplanes! Qua invece di mettere
                                  ; l'indirizzo lo prendiamo!!!

        TST.B         SuGiu                ; Dobbiamo salire o scendere? se SuGiu e'
                                  ; azzerata, (cioe' il TST verifica il BEQ)
                                  ; allora saltiamo a VAIGIU, se invece e' a $FF
                                  ; (se cioe' questo TST non e' verificato)
                                  ; continuiamo salendo (facendo dei sub)
        beq.w         VAIGIU
```

```
        cmp.l       #PIC-(40*30),d0        ; siamo arrivati abbastanza in ALTO?
        beq.s       MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        sub.l       #40,d0                ; sottraiamo 40, ossia 1 linea, facendo
                            ; scorrere in BASSO la figura
        bra.s       Finito

MettiGiu:
        clr.b       SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        bra.s       Finito                ; fara' saltare alla routine VAIGIU

VAIGIU:
        cmpi.l       #PIC+(40*30),d0        ; siamo arrivati abbastanza in BASSO?
        beq.s       MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        add.l       #40,d0                ; Aggiungiamo 40, ossia 1 linea, facendo
                            ; scorrere in ALTO la figura
        bra.s       finito

MettiSu:
        move.b       #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                        ; significa che dobbiamo risalire.

Finito:                                ; PUNTIAMO I PUNTATORI BITPLANES
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ       #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w       d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w       d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L       #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w       #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP2        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210        ; BPLCON0:
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                    ; 3 bitplanes lowres, non lace
BPLPOINTERS:
```

```
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

; Inserite qua il pezzo di copperlist

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

        dcb.b        40*30,0        ; questo spazio azzerato serve perche' spostandoci
                        ; a visualizzare piu' in basso e piu' in alto usciamo
                        ; dalla zona della PIC e visualizziamo quello che sta
                        ; prima e dopo la pic stessa, il che' causerebbe
                        ; la visualizzazione di byte sparsi di disturbo.
                        ; mettendo dei byte azzerati in quel punto viene
                        ; visualizzato $0000, ossia il colore di sfondo.

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        dcb.b        40*30,0        ; vedi sopra

; NOTA: Il dcb.b serve a mettere molti bytes uguali tra loro in memoria,
; scrivere dcb.b 10,0 e' come scrivere 10 volte dc.b 0.
;

        end
```

Questa routine in pratica aggiunge o sottrae 40 all'indirizzo a cui puntano i BPLPOINTERS in copperlist, leggendo per prima cosa l'indirizzo "attuale" con la routine opposta a quella che punta i bitplanes.
Con questo metodo si possono visualizzare anche immagini piu' grandi dello schermo, visualizzandone una parte per volta con la possibilita' di scorrerle in alto o in basso. Ad esempio nei giochi di FLIPPER, come PINBALL DREAMS, lo schermo di gioco e' piu' lungo di quello visibile, e scorre in alto o in basso, per visualizzare la parte dove rimbalza la pallina, cambiando i puntatori dei bitplanes.
In questo esempio spostandoci visualiziamo anche delle linee fuori dalla nostra figura, in quanto e' lunga 256 linee solamente e noi scorriamo di 30 linee sopra e 30 sotto, ossia 316 linne in totale. E' per questo che sono presenti dei dcb.b prima e dopo la figura, per "pulire" la zona che appare scorrendo fuori dai bitplanes RAW. Provate a cambiarli in questo modo:

```
        dcb.b        40*30,%11001100
```

Eseguendo il listato noterete che le parti fuori PIC sono a "STRISCE" anziche' azzerate, infatti le abbiamo riempite di %11001100110011001100110011001100
%11001100110011001100110011001100
%11001100110011001100110011001100
Ossia di strisce di bit.

```
Potete anche scorrere i 3 bitplanes sepatatamente: per fare cio' basta che
abilitiate 1 solo bitplane nel $dff100:

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane

E cambiate la posizione massima raggiungibile dallo scroll:

VAIGIU:
        cmpi.l        #PIC+(40*530),d0; siamo arrivati abbastanza in BASSO?
        beq.s        MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        ...

In questo modo vedrete scorrere i 3 bitplanes separatamente, infatti sono
posti l'uno dopo l'altro.

* Eccovi una modifica da fare alla copperlist: cosa succede se cambiamo tutti
gli 8 colori della figura ogni 2 linee? Copiate (con Amiga+b+c+i) questo pezzo
di copperlist ed inseritelo prima della fine della copperlist:

; Inserite qua il pezzo di copperlist

Cambiando la palette di 8 colori 52 volte, otterrete 8*52= 416 colori cambiati,
ma considerando che il color0, essendo lo sfondo, deve rimanere sempre NERO,
non viene modificato, solo gli altri 7, e non nell'ordine "numerico", ma in
ordine "sparso", infatti l'ordine con cui vengono aggiornati i colori non conta
sul risultato, si puo' cambiare prima il color2,poi il color3 ecc, mentre in
questo esempio "si parte" dal color5 ($dff18a), poi si cambia il color7 ecc.
Cambiando 7 colori 52 volte inserendo questa copperlist otteniamo 364 colori
effettivi sullo schermo contemporaneamente, il che non e' male, considerando
che lo schermo "ufficialmente" visualiziamo solo 8 colori. (7*52=364)


;2
        dc.w $18a,$102,$18e,$212,$182,$223        ; color5,color7,color2
        dc.w $18c,$323,$188,$323,$186,$334,$184,$434 ; col6,col4,col3,col2
        dc.w $5007,$fffe
;3
        dc.w $18a,$104,$18e,$214,$182,$225
        dc.w $18c,$324,$188,$324,$186,$335,$184,$435
        dc.w $5207,$fffe
;4
        dc.w $18a,$203,$18e,$313,$182,$324
        dc.w $18c,$423,$188,$423,$186,$434,$184,$534
        dc.w $5407,$fffe
;5
        dc.w $18a,$213,$18e,$313,$182,$324
        dc.w $18c,$433,$188,$433,$186,$434,$184,$534
        dc.w $5607,$fffe
;6
        dc.w $18a,$114,$18e,$214,$182,$224
        dc.w $18c,$323,$188,$323,$186,$334,$184,$434
        dc.w $5807,$fffe
;7
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$312,$188,$322,$186,$333,$184,$433
        dc.w $5a07,$fffe
;8
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$312,$188,$312,$186,$323,$184,$423
        dc.w $5c07,$fffe
```

```
;9
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$312,$188,$312,$186,$323,$184,$423
        dc.w $5e07,$fffe
;10
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$322,$188,$312,$186,$323,$184,$433
        dc.w $6007,$fffe
;11
        dc.w $18a,$110,$18e,$210,$182,$221
        dc.w $18c,$321,$188,$311,$186,$322,$184,$432
        dc.w $6207,$fffe
;12
        dc.w $18a,$210,$18e,$310,$182,$321
        dc.w $18c,$421,$188,$411,$186,$422,$184,$532
        dc.w $6407,$fffe
;13
        dc.w $18a,$210,$18e,$320,$182,$331
        dc.w $18c,$431,$188,$421,$186,$432,$184,$542
        dc.w $6607,$fffe
;14
        dc.w $18a,$220,$18e,$330,$182,$431
        dc.w $18c,$441,$188,$431,$186,$442,$184,$552
        dc.w $6807,$fffe
;15
        dc.w $18a,$220,$18e,$330,$182,$431
        dc.w $18c,$440,$188,$430,$186,$441,$184,$551
        dc.w $6a07,$fffe
;16
        dc.w $18a,$220,$18e,$330,$182,$431
        dc.w $18c,$441,$188,$431,$186,$442,$184,$552
        dc.w $6c07,$fffe
;17
        dc.w $18a,$120,$18e,$230,$182,$331
        dc.w $18c,$341,$188,$331,$186,$342,$184,$452
        dc.w $6e07,$fffe
;18
        dc.w $18a,$120,$18e,$230,$182,$341
        dc.w $18c,$351,$188,$341,$186,$352,$184,$462
        dc.w $7007,$fffe
;19
        dc.w $18a,$121,$18e,$231,$182,$332
        dc.w $18c,$342,$188,$332,$186,$343,$184,$453
        dc.w $7207,$fffe
;20
        dc.w $18a,$021,$18e,$131,$182,$232
        dc.w $18c,$242,$188,$232,$186,$243,$184,$353
        dc.w $7407,$fffe
;21
        dc.w $18a,$022,$18e,$132,$182,$233
        dc.w $18c,$243,$188,$233,$186,$244,$184,$354
        dc.w $7607,$fffe
;22
        dc.w $18a,$012,$18e,$122,$182,$223
        dc.w $18c,$233,$188,$223,$186,$234,$184,$344
        dc.w $7807,$fffe
;23
        dc.w $18a,$013,$18e,$123,$182,$224
        dc.w $18c,$234,$188,$224,$186,$235,$184,$345
        dc.w $7a07,$fffe
;24
        dc.w $18a,$013,$18e,$023,$182,$124
```

```
        dc.w $18c,$134,$188,$124,$186,$135,$184,$245
        dc.w $7c07,$fffe
;25
        dc.w $18a,$013,$18e,$123,$182,$224
        dc.w $18c,$234,$188,$224,$186,$235,$184,$345
        dc.w $7e07,$fffe
;26
        dc.w $18a,$012,$18e,$122,$182,$223
        dc.w $18c,$233,$188,$223,$186,$234,$184,$344
        dc.w $8007,$fffe
;27
        dc.w $18a,$022,$18e,$132,$182,$233
        dc.w $18c,$243,$188,$233,$186,$244,$184,$354
        dc.w $8207,$fffe
;28
        dc.w $18a,$112,$18e,$132,$182,$233
        dc.w $18c,$233,$188,$233,$186,$244,$184,$344
        dc.w $8407,$fffe
;29
        dc.w $18a,$102,$18e,$222,$182,$223
        dc.w $18c,$323,$188,$323,$186,$334,$184,$443
        dc.w $8607,$fffe
;30
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$322,$188,$322,$186,$333,$184,$433
        dc.w $8807,$fffe
;31
        dc.w $18a,$104,$18e,$214,$182,$225
        dc.w $18c,$324,$188,$324,$186,$335,$184,$435
        dc.w $8a07,$fffe
;32
        dc.w $18a,$203,$18e,$313,$182,$324
        dc.w $18c,$423,$188,$423,$186,$434,$184,$534
        dc.w $8c07,$fffe
;33
        dc.w $18a,$213,$18e,$313,$182,$324
        dc.w $18c,$433,$188,$433,$186,$434,$184,$534
        dc.w $8e07,$fffe
;34
        dc.w $18a,$114,$18e,$214,$182,$224
        dc.w $18c,$323,$188,$323,$186,$334,$184,$434
        dc.w $9007,$fffe
;35
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$312,$188,$322,$186,$333,$184,$433
        dc.w $9207,$fffe
;36
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$312,$188,$312,$186,$323,$184,$423
        dc.w $9407,$fffe
;37
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$312,$188,$312,$186,$323,$184,$423
        dc.w $9607,$fffe
;38
        dc.w $18a,$101,$18e,$211,$182,$222
        dc.w $18c,$322,$188,$312,$186,$323,$184,$433
        dc.w $9807,$fffe
;39
        dc.w $18a,$110,$18e,$210,$182,$221
        dc.w $18c,$321,$188,$311,$186,$322,$184,$432
        dc.w $9a07,$fffe
```

```
;40
        dc.w $18a,$210,$18e,$310,$182,$321
        dc.w $18c,$421,$188,$411,$186,$422,$184,$532
        dc.w $9c07,$fffe
;41
        dc.w $18a,$210,$18e,$320,$182,$331
        dc.w $18c,$431,$188,$421,$186,$432,$184,$542
        dc.w $9e07,$fffe
;42
        dc.w $18a,$220,$18e,$330,$182,$431
        dc.w $18c,$441,$188,$431,$186,$442,$184,$552
        dc.w $a007,$fffe
;43
        dc.w $18a,$220,$18e,$330,$182,$431
        dc.w $18c,$440,$188,$430,$186,$441,$184,$551
        dc.w $a207,$fffe
;44
        dc.w $18a,$220,$18e,$330,$182,$431
        dc.w $18c,$441,$188,$431,$186,$442,$184,$552
        dc.w $a407,$fffe
;45
        dc.w $18a,$120,$18e,$230,$182,$331
        dc.w $18c,$341,$188,$331,$186,$342,$184,$452
        dc.w $a607,$fffe
;46
        dc.w $18a,$120,$18e,$230,$182,$341
        dc.w $18c,$351,$188,$341,$186,$352,$184,$462
        dc.w $a807,$fffe
;47
        dc.w $18a,$121,$18e,$231,$182,$332
        dc.w $18c,$342,$188,$332,$186,$343,$184,$453
        dc.w $aa07,$fffe
;48
        dc.w $18a,$021,$18e,$131,$182,$232
        dc.w $18c,$242,$188,$232,$186,$243,$184,$353
        dc.w $ac07,$fffe
;49
        dc.w $18a,$022,$18e,$132,$182,$233
        dc.w $18c,$243,$188,$233,$186,$244,$184,$354
        dc.w $ae07,$fffe
;50
        dc.w $18a,$012,$18e,$122,$182,$223
        dc.w $18c,$233,$188,$223,$186,$234,$184,$344
        dc.w $b007,$fffe
;51
        dc.w $18a,$013,$18e,$123,$182,$224
        dc.w $18c,$234,$188,$224,$186,$235,$184,$345
        dc.w $b207,$fffe
;52
        dc.w $18a,$013,$18e,$023,$182,$124
        dc.w $18c,$134,$188,$124,$186,$135,$184,$245
        dc.w $b407,$fffe
;53
        dc.w $18a,$013,$18e,$123,$182,$224
        dc.w $18c,$234,$188,$224,$186,$235,$184,$345
        dc.w $b607,$fffe
;54
        dc.w $18a,$012,$18e,$122,$182,$223
        dc.w $18c,$233,$188,$223,$186,$234,$184,$344
```

21.4 Lezione5d

```
; Lezione5d.s        SCORRIMENTO DI UNA FIGURA IN ALTO E IN BASSO MODIFICANDO I
;                PUNTATORI AI PITPLANES NELLA COPPERLIST PIU' SCORRIMENTO
;                A DESTRA E SINISTRA TRAMITE IL $dff102 (BPLCON0)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti


        bsr.w        MuoviCopper        ; fa scorrere la figura in alto e in basso
                                ; di una linea alla volta cambiando i
                                ; puntatori ai bitplanes in copperlist

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w        MuoviCopper2        ; fa scorrere col $dff102 la figura a destra
                                ; e a sinistra (massimo 15 pixel)

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop
```

```
        move.l        4.w,a6
        jsr          -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr          -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                           ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l         0        ; della graphics.library

OldCop:                          ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l         0


;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)
;        La struttura e' simile a quella di Lezione3d.s

MuoviCopper:
        LEA          BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w         2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap          d0               ; attualmente il $dff0e0 e lo poniamo
        move.w         6(a1),d0        ; in d0
        TST.B          SuGiu               ; Dobbiamo salire o scendere?
        beq.w          VAIGIU
        cmp.l          #PIC-(40*30),d0        ; siamo arrivati abbastanza in ALTO?
        beq.s          MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        sub.l          #40,d0               ; sottraiamo 40, ossia 1 linea
        bra.s          Finito

MettiGiu:
        clr.b          SuGiu               ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        bra.s          Finito               ; fara' saltare alla routine VAIGIU

VAIGIU:
        cmpi.l         #PIC+(40*30),d0        ; siamo arrivati abbastanza in BASSO?
        beq.s          MettiSu               ; se si, siamo in fondo e dobbiamo risalire
        add.l          #40,d0               ; Aggiungiamo 40, ossia 1 linea
        bra.s          finito

MettiSu:
        move.b         #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                          ; significa che dobbiamo risalire.

Finito:                                ; PUNTIAMO I PUNTATORI BITPLANES
        LEA          BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ          #2,D1               ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w         d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap          d0               ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w         d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap          d0               ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L          #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w         #8,a1               ; andiamo ai prossimi bplpointers nella COP
        dbra          d1,POINTBP2        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts
```

```
;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b        0,0

;*****************************************************************************
MuoviCopper2:
        TST.B       FLAG                ; Dobbiamo avanzare o indietreggiare?
        beq.w       AVANTI
        cmpi.b       #$00,MIOCON1        ; siamo arrivati alla posizione normale?
        beq.s       MettiAvanti        ; se si, dobbiamo avanzare!
        sub.b       #$11,MIOCON1        ; sottraiamo 1 allo scroll dei bitplanes
        rts

MettiAvanti:
        clr.b       FLAG                ; Azzerando FLAG, al TST.B FLAG il BEQ
        rts                         ; fara' saltare alla routine AVANTI

AVANTI:
        cmpi.b       #$ff,MIOCON1        ; siamo arrivati allo scroll massimo in
                                    ; avanti, ossia $FF? ($f pari e $f dispari)
        beq.s       MettiIndietro       ; se si, siamo dobbiamo tornare indietro
        add.b       #$11,MIOCON1        ; aggiungiamo 1 allo scroll dei bitplanes
        rts

MettiIndietro:
        move.b       #$ff,FLAG        ; Quando la label FLAG non e' a zero,
        rts                         ; significa che dobbiamo indietreggiare

;        Questo byte e' un FLAG, ossia serve per indicare se andare avanti o
;        indietro.

FLAG:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop

        dc.w        $102                ; BplCon1 - IL REGISTRO
        dc.b        $00                ; BplCon1 - IL BYTE NON UTILIZZATO!!!
MIOCON1:
        dc.b        $00                ; BplCon1 - IL BYTE UTILIZZATO!!!

        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210        ; BPLCON0:
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                    ; 3 bitplanes lowres, non lace
BPLPOINTERS:
```

```
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane - BPL0PT
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane - BPL1PT
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane - BPL2PT

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7
        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

        dcb.b        40*30,0                        ; spazio azzerato

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        dcb.b        40*30,0                        ; spazio azzerato

        end
```

Nulla di nuovo, semplicemente sono stati uniti i sorgenti precedenti della Lezione5.s

Lezione5d2

```
; Lezione5d2.s        SCORRIMENTO DI UNA FIGURA IN ALTO E IN BASSO MODIFICANDO I
;                PUNTATORI AI PITPLANES NELLA COPPERLIST + EFFETTO DISTORSIONE
;                OTTENUTO CON I $dff102 (bplcon1)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
```

```
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                 ; Facciamo partire la COP
        move.w        #0,$dff1fc                 ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s         mouse               ; Se non ancora, non andare avanti

        btst         #2,$dff016         ; se il tasto destro e' premuto salta
        beq.s         Aspetta                ; la routine dello scroll, bloccandolo


        bsr.w         MuoviCopper         ; fa scorrere la figura in alto e in basso
                                  ; di una linea alla volta cambiando i
                                  ; puntatori ai bitplanes in copperlist

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s         Aspetta               ; Se si, non andare avanti, aspetta!

        btst         #6,$bfe001         ; tasto sinistro del mouse premuto?
        bne.s         mouse                 ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)         ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1         ; Base della libreria da chiudere
        jsr         -$19e(a6)         ; Closelibrary - chiudo la graphics lib
        rts                          ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l         0         ; della graphics.library

OldCop:                          ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l         0


;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)
;        La struttura e' simile a quella di Lezione3d.s

MuoviCopper:
        LEA         BPLPOINTERS,A1         ; Con queste 4 istruzioni preleviamo dalla
        move.w         2(a1),d0         ; copperlist l'indirizzo dove sta puntando
        swap         d0                 ; attualmente il $dff0e0 e lo poniamo
        move.w         6(a1),d0         ; in d0 - il contrario della routine che
                                  ; punta i bitplanes! Qua invece di mettere
                                  ; l'indirizzo lo prendiamo!!!

        TST.B         SuGiu                 ; Dobbiamo salire o scendere? se SuGiu e'
                                  ; azzerata, (cioe' il TST verifica il BEQ)
                                  ; allora saltiamo a VAIGIU, se invece e' a $FF
                                  ; (se cioe' questo TST non e' verificato)
```

```
                            ; continuiamo salendo (facendo dei sub)
        beq.w       VAIGIU
        cmp.l       #PIC-(40*30),d0        ; siamo arrivati abbastanza in BASSO?
        beq.s       MettiGiu        ; se si, siamo in fondo e dobbiamo risalire
        sub.l       #40,d0                ; sottraiamo 40, ossia 1 linea, facendo
                            ; scorrere in BASSO la figura
        bra.s       Finito

MettiGiu:
        clr.b       SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        bra.s       Finito                ; fara' saltare alla routine VAIGIU

VAIGIU:
        cmpi.l       #PIC+(40*30),d0        ; siamo arrivati abbastanza in ALTO?
        beq.s       MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        add.l       #40,d0                ; Aggiungiamo 40, ossia 1 linea, facendo
                            ; scorrere in ALTO la figura
        bra.s       finito

MettiSu:
        move.b       #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                        ; significa che dobbiamo risalire.

Finito:                                ; PUNTIAMO I PUNTATORI BITPLANES
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP2        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
```

```
                                        ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo          bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo          bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

;        EFFETTO SPECCHIO (che si potrebbe vendere per effetto "texturemap")

        dc.w        $7007,$fffe
        dc.w        $180,$004        ; Color0
        dc.w        $102,$011        ; bplcon1
        dc.w        $7307,$fffe
        dc.w        $180,$006        ; Color0
        dc.w        $102,$022        ; bplcon1
        dc.w        $7607,$fffe
        dc.w        $180,$008        ; Color0
        dc.w        $102,$033        ; bplcon1
        dc.w        $7b07,$fffe
        dc.w        $180,$00a        ; Color0
        dc.w        $102,$044        ; bplcon1
        dc.w        $8307,$fffe
        dc.w        $180,$00c        ; Color0
        dc.w        $102,$055        ; bplcon1
        dc.w        $9007,$fffe
        dc.w        $180,$00e        ; Color0
        dc.w        $102,$066        ; bplcon1
        dc.w        $9607,$fffe
        dc.w        $180,$00f        ; Color0
        dc.w        $102,$077        ; bplcon1
        dc.w        $9a07,$fffe
        dc.w        $180,$00e        ; Color0
        dc.w        $a007,$fffe
        dc.w        $180,$00c        ; Color0
        dc.w        $102,$066        ; bplcon1
        dc.w        $ad07,$fffe
        dc.w        $180,$00a        ; Color0
        dc.w        $102,$055        ; bplcon1
        dc.w        $b507,$fffe
        dc.w        $180,$008        ; Color0
        dc.w        $102,$044        ; bplcon1
        dc.w        $ba07,$fffe
        dc.w        $180,$006        ; Color0
        dc.w        $102,$033        ; bplcon1
        dc.w        $bd07,$fffe
        dc.w        $180,$004        ; Color0
        dc.w        $102,$022        ; bplcon1
        dc.w        $bf07,$fffe
        dc.w        $180,$001        ; Color0

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura
```

```
        dcb.b        40*98,0               ; spazio azzerato

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                       ; convertita col KEFCON, fatta di
                                       ; 3 bitplanes consecutivi

        dcb.b        40*30,0               ; spazio azzerato

        end
```

cambiando la sola copperlist dell'esempio Lezione5c.s si e' potuto ottenere questo effetto di "avvolgimento della figura su un cilindro", che non e' poi molto convincente, ma almeno e' molto facile e veloce da fare, infatti basta mettere i $dff102 in ordine progressivo nei wait: 1,2,3,4 per creare la prima distorsione verso destra:

```
        ++++++++++++
         ++++++++++++
          ++++++++++++
           ++++++++++++
```

poi raggiunta la meta' basta calare di uno ogni volta fino a tornare a zero.

21.5 Lezione5e

```
; Lezione5e.s        DIMEZZAMENTO DELL'ALTEZZA DI UNA FIGURA MODIFICANDO I MODULI

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
```

```
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti
frame:
        cmpi.b        #$fe,$dff006        ; Siamo alla linea 254? (deve rifare il giro!)
        bne.s        frame                ; Se non ancora, non andare avanti
frame2:
        cmpi.b        #$fd,$dff006        ; Siamo alla linea 253?
        bne.s        frame2                ; Se non ancora, non andare avanti
frame3:
        cmpi.b        #$fc,$dff006        ; Siamo alla linea 252?
        bne.s        frame3
frame4:
        cmpi.b        #$fb,$dff006        ; Siamo alla linea 251?
        bne.s        frame4

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        NonMuovere        ; la routine dello scroll, bloccandolo

        bsr.s        MuoviCopper        ; Routine dei moduli

NonMuovere
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

; Con questa routine aggiungiamo o sottraiamo 40 ai registri dei moduli per
; accorciare la figura. Ho mantenuto le label dell'esempio precedente per
; risparmiare tempo.

MuoviCopper:
        TST.B        SuGiu                ; Dobbiamo salire o scendere?
        beq.w        VAIGIU
        tst.w        MOD1                ; Siamo al valore normale del modulo? (ZERO)
        beq.s        MettiGiu        ; se si, dobbiamo aumentare il valore
        sub.w        #40,MOD1        ; sottraiamo 40, ossia 1 linea, facendo
                                ; scorrere in BASSO la figura (ingrandendola)
        sub.w        #40,MOD2        ; sottraiamo 40 al modulo 2
        rts

MettiGiu:
        clr.b        SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                        ; fara' saltare alla routine VAIGIU
```

```
VAIGIU:
        cmpi.w        #40*20,MOD1        ; abbiamo dimezzato abbastanza??
        beq.s         MettiSu               ; se si, dobbiamo far tornare la pic normale
        add.w         #40,MOD1        ; Aggiungiamo 40, ossia 1 linea, facendo
                                ; scorrere in ALTO la figura (dimezzandola)
        add.w         #40,MOD2        ; Aggiungiamo 40 al modulo 2
        rts

MettiSu:
        move.b        #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                         ; significa che dobbiamo risalire.


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108
MOD1:
        dc.w        0                ; Bpl1Mod
        dc.w        $10a
MOD2:
        DC.W        0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                    ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7
        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura


PIC:
```

```
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                       ; convertita col KEFCON, fatta di
                                       ; 3 bitplanes consecutivi
        end
```

Per fare una cosa piu' "pulita" avrei dovuto mettere un wait sotto la figura
che mi togliesse di mezzo le "impurita" che si vedono in basso, ossia i byte
in memoria dopo la figura e i bitplanes che "spuntano" sotto il primo e il
secondo. Ma lo scopo principale e' quello di spiegare la funzione dei moduli.
La routine tra l'altro viene eseguita una volta ogni 4 fotogrammi per
rallentarla.

21.6 Lezione5f

```
; Lezione5f.s        EFFETTO "SCIOGLIMENTO" O "FLOOD" FATTO CON I MODULI NEGATIVI

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w        Flood                ; Muove in alto e in basso un wait seguito
                                ; da un modulo -40, che causa l'effetto FLOOD

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!
```

```
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

; Effetto definibile come "Metallo fuso", ottenuto con i moduli -40

Flood:
        TST.B        SuGiu                ; Dobbiamo salire o scendere?
        beq.w        VAIGIU
        cmp.b        #$30,FWAIT        ; siamo arrivati abbastanza in ALTO?
        beq.s        MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        subq.b        #1,FWAIT        ; scorriamo in ALTO
        rts

MettiGiu:
        clr.b        SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts

VAIGIU:
        cmp.b        #$f0,FWAIT        ; siamo arrivati abbastanza in BASSO?
        beq.s        MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        addq.b        #1,FWAIT        ; scorriamo in ALTO
        rts

MettiSu:
        move.b        #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                        ; significa che dobbiamo risalire.


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b        0,0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000
```

```
        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                        ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

FWAIT:
        dc.w        $3007,$FFFE        ; WAIT che precede il modulo negativo
        dc.w        $108,-40
        dc.w        $10a,-40

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        end
```

Da notare che -40 viene assemblato come $ffd8 (provate con un "?-40").
Provate a bloccare col tasto destro la routine e verificherete che avviene un
"allungamento" dell'ultima riga fino alla fine dello schermo.
Abbiamo verificato che con un modulo negativo di -40 il copper non avanza,
infatti va avanti di 40 e torna indietro di 40. Ma se mettiamo il modulo a -80,
cosa succede??? Legge alla rovescia!!! infatti legge e visualizza 40 bytes, poi
indietreggia di 80 bytes, andando all'inizio della linea precedente, che viene
visualizzata, dopodiche' salta alla linea precedente eccetera. Questo sistema
e' il piu' usato per gli effetti SPECCHIO tanto frequenti sull'Amiga proprio
perche' basta mettere un paio di istruzioni copper:

```
        dc.w        $108,-80
        dc.w        $10a,-80
```

provate a cambiare i due -40 dei moduli in questo esempio in due -80, e lo
"SPECCHIO" apparira', anche se stavolta il problema e' che viene visualizzato
anche qualcosa che sta sopra la figura (procedendo all'indietro).
Una curiosita': noterete che nella prima linea dello "SPORCO" che appare dopo
la figura specchiata c'e' un movimento che interessa dei pixel: si tratta del
wait nella copperlist che cambiamo ogni frame! Infatti cosa c'e' in memoria

```
prima della nostra figura?? La copperlist!! Dunque procedendo all'indietro
nella lettura (modulo -80), cosa sara' visualizzato?? I byte della copperlist,
poi quello che viene prima.

Se aumentiamo la negativita' otterremo specchiature sempre piu' schiacciate,
infatti avviene lo stesso effetto dei moduli positivi, ma al rovescio.

        dc.w        $108,-40*3
        dc.w        $10a,-40*3

Per la figura rispecchiata dimezzata, eccetera.
```

21.7 Lezione5g

```
; Lezione5g.s        SCORRIMENTO DI UNA FIGURA IN ALTO E IN BASSO MODIFICANDO I
;                PUNTATORI AI PITPLANES NELLA COPPERLIST + EFFETTO SPECCHIO
;                OTTENUTO CON I MODULI NEGATIVI (-40*2, -40*3, -40*4...)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr          -$78(a6)        ; Disable - ferma il multitasking
        lea          GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr          -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA          BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ          #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst         #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s         Aspetta                ; la routine dello scroll, bloccandolo


        bsr.w         MuoviCopper        ; fa scorrere la figura in alto e in basso
                                ; di una linea alla volta cambiando i
                                ; puntatori ai bitplanes in copperlist
```

```
Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s         Aspetta               ; Se si, non andare avanti, aspetta!

        btst         #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s         mouse                ; se no, torna a mouse:

        move.l         OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w         d0,$dff088                ; facciamo partire la vecchia cop

        move.l         4.w,a6
        jsr         -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l         gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr         -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                         ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l         0        ; della graphics.library

OldCop:                          ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l         0


;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)
;        La struttura e' simile a quella di Lezione3d.s

MuoviCopper:
        LEA         BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w         2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap         d0                ; attualmente il $dff0e0 e lo poniamo
        move.w         6(a1),d0        ; in d0 - il contrario della routine che
                                ; punta i bitplanes! Qua invece di mettere
                                ; l'indirizzo lo prendiamo!!!

        TST.B         SuGiu                ; Dobbiamo salire o scendere? se SuGiu e'
                                ; azzerata, (cioe' il TST verifica il BEQ)
                                ; allora saltiamo a VAIGIU, se invece e' a $FF
                                ; (se cioe' questo TST non e' verificato)
                                ; continuiamo salendo (facendo dei sub)
        beq.w         VAIGIU
        cmp.l         #PIC-(40*18),d0        ; siamo arrivati abbastanza in BASSO?
        beq.s         MettiGiu        ; se si, siamo in fondo e dobbiamo risalire
        sub.l         #40,d0                ; sottraiamo 40, ossia 1 linea, facendo
                                ; scorrere in BASSO la figura
        bra.s         Finito

MettiGiu:
        clr.b         SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        bra.s         Finito                ; fara' saltare alla routine VAIGIU

VAIGIU:
        cmpi.l         #PIC+(40*130),d0        ; siamo arrivati abbastanza in ALTO?
        beq.s         MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        add.l         #40,d0                ; Aggiungiamo 40, ossia 1 linea, facendo
                                ; scorrere in ALTO la figura
        bra.s         finito
```

```
MettiSu:
        move.b        #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                          ; significa che dobbiamo risalire.

Finito:                                 ; PUNTIAMO I PUNTATORI BITPLANES
        LEA         BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ         #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w        d0,6(a1)          ; copia la word BASSA dell'indirizzo del plane
        swap         d0                 ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)          ; copia la word ALTA dell'indirizzo del plane
        swap         d0                 ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0         ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                 ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP2         ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b         0,0


        SECTION         GRAPHIC,DATA_C

COPPERLIST:
        dc.w         $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w         $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w         $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w         $13e,$0000

        dc.w         $8e,$2c81         ; DiwStrt        (registri con valori normali)
        dc.w         $90,$2cc1         ; DiwStop
        dc.w         $92,$0038         ; DdfStart
        dc.w         $94,$00d0         ; DdfStop
        dc.w         $102,0                 ; BplCon1
        dc.w         $104,0                 ; BplCon2
        dc.w         $108,0                 ; Bpl1Mod
        dc.w         $10a,0                 ; Bpl2Mod

                     ; 5432109876543210
        dc.w         $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                      ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000         ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000         ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000         ;terzo         bitplane

        dc.w         $0180,$000         ; color0
        dc.w         $0182,$475         ; color1
        dc.w         $0184,$fff         ; color2
        dc.w         $0186,$ccc         ; color3
        dc.w         $0188,$999         ; color4
        dc.w         $018a,$232         ; color5
        dc.w         $018c,$777         ; color6
        dc.w         $018e,$444         ; color7

;        EFFETTO SPECCHIO (che si potrebbe vendere per effetto "texturemap")

        dc.w         $b007,$fffe
```

```
        dc.w        $180,$004       ; Color0
        dc.w        $108,-40*7       ; Bpl1Mod - specchio dimezzato 5 volte
        dc.w        $10a,-40*7       ; Bpl2Mod
        dc.w        $b307,$fffe
        dc.w        $180,$006       ; Color0
        dc.w        $108,-40*6       ; Bpl1Mod - specchio dimezzato 4 volte
        dc.w        $10a,-40*6       ; Bpl2Mod
        dc.w        $b607,$fffe
        dc.w        $180,$008       ; Color0
        dc.w        $108,-40*5       ; Bpl1Mod - specchio dimezzato 3 volte
        dc.w        $10a,-40*5       ; Bpl2Mod
        dc.w        $bb07,$fffe
        dc.w        $180,$00a       ; Color0
        dc.w        $108,-40*4       ; Bpl1Mod - specchio dimezzato 2 volte
        dc.w        $10a,-40*4       ; Bpl2Mod
        dc.w        $c307,$fffe
        dc.w        $180,$00c       ; Color0
        dc.w        $108,-40*3       ; Bpl1Mod - specchio dimezzato
        dc.w        $10a,-40*3       ; Bpl2Mod
        dc.w        $d007,$fffe
        dc.w        $180,$00e       ; Color0
        dc.w        $108,-40*2       ; Bpl1Mod - specchio normale
        dc.w        $10a,-40*2       ; Bpl2Mod
        dc.w        $d607,$fffe
        dc.w        $180,$00f       ; Color0
        dc.w        $108,-40       ; Bpl1Mod - FLOOD, linee ripetute per
        dc.w        $10a,-40       ; Bpl2Mod - effetto centrale di ingrandimento
        dc.w        $da07,$fffe
        dc.w        $180,$00e       ; Color0
        dc.w        $108,-40*2       ; Bpl1Mod - specchio normale
        dc.w        $10a,-40*2       ; Bpl2Mod
        dc.w        $e007,$fffe
        dc.w        $180,$00c       ; Color0
        dc.w        $108,-40*3       ; Bpl1Mod - specchio dimezzato
        dc.w        $10a,-40*3       ; Bpl2Mod
        dc.w        $ed07,$fffe
        dc.w        $180,$00a       ; Color0
        dc.w        $108,-40*4       ; Bpl1Mod - specchio dimezzato 2 volte
        dc.w        $10a,-40*4       ; Bpl2Mod
        dc.w        $f507,$fffe
        dc.w        $180,$008       ; Color0
        dc.w        $108,-40*5       ; Bpl1Mod - specchio dimezzato 3 volte
        dc.w        $10a,-40*5       ; Bpl2Mod
        dc.w        $fa07,$fffe
        dc.w        $180,$006       ; Color0
        dc.w        $108,-40*6       ; Bpl1Mod - specchio dimezzato 4 volte
        dc.w        $10a,-40*6       ; Bpl2Mod
        dc.w        $fd07,$fffe
        dc.w        $180,$004       ; Color0
        dc.w        $108,-40*7       ; Bpl1Mod - specchio dimezzato 5 volte
        dc.w        $10a,-40*7       ; Bpl2Mod
        dc.w        $ff07,$fffe
        dc.w        $180,$002       ; Color0
        dc.w        $108,-40       ; ferma l'immagine per evitare di visualizzare
        dc.w        $10a,-40       ; i byte prima della RAW

        dc.w        $FFFF,$FFFE       ; Fine della copperlist

;        figura

        dcb.b       40*98,0              ; spazio azzerato
```

```
PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                               ; convertita col KEFCON, fatta di
                                               ; 3 bitplanes consecutivi

        dcb.b        40*30,0                ; spazio azzerato

        end
```

In questo esempio mettendo dei moduli negativi per creare specchiature sempre piu' "DIMEZZATE", e' stato possibile simulare "un'avvolgimento" dell'immagine specchiata su una superficie rozzamente "curva". Disponendo bene i moduli si possono generare effetti del tipo ZOOM o LENTE DI INGRANDIMENTO, nonche' di distorsione cilindrica come questo esempio, specialmente se si aiuta l'effetto ottico con i colori (in questo caso con una tonalita' di blu).
Il sorgente e' lo stesso di Lezione5c.s, l'unica modifica e' nella copperlist. Per aggiungere realismo all'effetto di "avvolgimento su un cilindro" si puo' simulare una curvatura con i $dff102 (bplcon1), come gia' visto nel listato Lezione5d2.s. Sostituite la copperlist dell'esempio con questa, che e' una fusione con quella di Lezione5d2.s.

- Ricordo che per rimuovere la vecchia parte di copperlist potete usare l'opzione Amiga+b per selezionarla e Amiga+x per il taglio a selezione fatta, mentre per copiare questa parte di copperlist sopra, selezionatela con Amiga+b, poi Amiga+c per copiare, posizionatevi nel punto giusto in copperlist e inseritela con Amiga+i.

```
        dc.w        $b007,$fffe
        dc.w        $180,$004        ; Color0
        dc.w        $102,$011        ; bplcon1
        dc.w        $108,-40*7        ; Bpl1Mod - specchio dimezzato 5 volte
        dc.w        $10a,-40*7        ; Bpl2Mod
        dc.w        $b307,$fffe
        dc.w        $180,$006        ; Color0
        dc.w        $102,$022        ; bplcon1
        dc.w        $108,-40*6        ; Bpl1Mod - specchio dimezzato 4 volte
        dc.w        $10a,-40*6        ; Bpl2Mod
        dc.w        $b607,$fffe
        dc.w        $180,$008        ; Color0
        dc.w        $102,$033        ; bplcon1
        dc.w        $108,-40*5        ; Bpl1Mod - specchio dimezzato 3 volte
        dc.w        $10a,-40*5        ; Bpl2Mod
        dc.w        $bb07,$fffe
        dc.w        $180,$00a        ; Color0
        dc.w        $102,$044        ; bplcon1
        dc.w        $108,-40*4        ; Bpl1Mod - specchio dimezzato 2 volte
        dc.w        $10a,-40*4        ; Bpl2Mod
        dc.w        $c307,$fffe
        dc.w        $180,$00c        ; Color0
        dc.w        $102,$055        ; bplcon1
        dc.w        $108,-40*3        ; Bpl1Mod - specchio dimezzato
        dc.w        $10a,-40*3        ; Bpl2Mod
        dc.w        $d007,$fffe
        dc.w        $180,$00e        ; Color0
        dc.w        $102,$066        ; bplcon1
        dc.w        $108,-40*2        ; Bpl1Mod - specchio normale
        dc.w        $10a,-40*2        ; Bpl2Mod
        dc.w        $d607,$fffe
        dc.w        $180,$00f        ; Color0
        dc.w        $102,$077        ; bplcon1
        dc.w        $108,-40        ; Bpl1Mod - FLOOD, linee ripetute per
```

```
        dc.w        $10a,-40        ; Bpl2Mod - effetto centrale di ingrandimento
        dc.w        $da07,$fffe
        dc.w        $180,$00e        ; Color0
        dc.w        $102,$066        ; bplcon1
        dc.w        $108,-40*2        ; Bpl1Mod - specchio normale
        dc.w        $10a,-40*2        ; Bpl2Mod
        dc.w        $e007,$fffe
        dc.w        $180,$00c        ; Color0
        dc.w        $102,$055        ; bplcon1
        dc.w        $108,-40*3        ; Bpl1Mod - specchio dimezzato
        dc.w        $10a,-40*3        ; Bpl2Mod
        dc.w        $ed07,$fffe
        dc.w        $180,$00a        ; Color0
        dc.w        $102,$044        ; bplcon1
        dc.w        $108,-40*4        ; Bpl1Mod - specchio dimezzato 2 volte
        dc.w        $10a,-40*4        ; Bpl2Mod
        dc.w        $f507,$fffe
        dc.w        $180,$008        ; Color0
        dc.w        $102,$033        ; bplcon1
        dc.w        $108,-40*5        ; Bpl1Mod - specchio dimezzato 3 volte
        dc.w        $10a,-40*5        ; Bpl2Mod
        dc.w        $fa07,$fffe
        dc.w        $180,$006        ; Color0
        dc.w        $102,$022        ; bplcon1
        dc.w        $108,-40*6        ; Bpl1Mod - specchio dimezzato 4 volte
        dc.w        $10a,-40*6        ; Bpl2Mod
        dc.w        $fd07,$fffe
        dc.w        $180,$004        ; Color0
        dc.w        $102,$011        ; bplcon1
        dc.w        $108,-40*7        ; Bpl1Mod - specchio dimezzato 5 volte
        dc.w        $10a,-40*7        ; Bpl2Mod
        dc.w        $ff07,$fffe
        dc.w        $180,$002        ; Color0
        dc.w        $102,$000        ; bplcon1
        dc.w        $108,-40        ; ferma l'immagine per evitare di visualizzare
        dc.w        $10a,-40        ; i byte prima della RAW
```

21.8 Lezione5h

```
; Lezione5h.s        ONDULAZIONE ORIZZONTALE DI UNA FIGURA COL $dff102

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
```

```
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w        Ondula                ; fa ondulare la figura con molti $dff102 in
                                ; copperlist

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

; Questa routine e' simile a quella in Lezione3e.s, infatti vengono "spostati"
; dei valori come in una catena; vi ricordate il sistema gia' usato:
;
;        move.w        col2,col1        ; col2 copiato in col1
;        move.w        col3,col2        ; col3 copiato in col2
;        move.w        col4,col3        ; col4 copiato in col3
;        move.w        col5,col4        ; col5 copiato in col4
;
; In questa routine invece di copiare colori si copiano valori del $dff102, ma
; il funzionamento della routine e' lo stesso. Per risparmiare LABEL e tempo
; la routine e' stata fornita di un ciclo DBRA che esegue la rotazione di
```

```
; quante word vogliamo: essendo le word da cambiare distanti 8 bytes, basta
; mettere l'indirizzo di una in a0 e dell'altra in a1 e lo spostamento avviene
; con un MOVE.W (a0),(a1). Poi passiamo alla coppia seguente aggiungendo 8
; ad a0 e a1, che punteranno alla prossima coppia di word da scambiare.
; Ricorderete che per fare il ciclo INFINITO bisogna che il primo valore sia
; sempre rimpiazzato dall'ultimo:
;
;         >>>>>>>>>>>>>>>>>>>>
;        ^                     v
; In questo caso al termine del ciclo viene copiato il primo valore nell'ultimo
; per cui l'afflusso e' costante; la vecchia routine infatti terminava cosi':
;
;        move.w        col1,col14        ; col1 copiato in col14
;

Ondula:
        LEA         CON1EFFETTO+8,A0 ; Indirizzo word sorgente in a0
        LEA         CON1EFFETTO,A1        ; Indirizzo delle word destinazione in a1
        MOVEQ         #44,D2                ; 45 bplcon1 da cambiare in COPLIST
SCAMBIA:
        MOVE.W         (A0),(A1)        ; copia due word consecutive - scorrimento!
        ADDQ.W         #8,A0                ; prossima coppia di word
        ADDQ.W         #8,A1                ; prossima coppia di word
        DBRA         D2,SCAMBIA        ; ripeti "SCAMBIA" il numero giusto di VOLTE

        MOVE.W         CON1EFFETTO,ULTIMOVALORE ; per rendere infinito il ciclo
        RTS                                 ; copiamo il primo valore nell'ultimo
                                          ; ogni volta.


        SECTION         GRAPHIC,DATA_C

COPPERLIST:
        dc.w         $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w         $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w         $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w         $13e,$0000

        dc.w         $8e,$2c81         ; DiwStrt        (registri con valori normali)
        dc.w         $90,$2cc1         ; DiwStop
        dc.w         $92,$0038         ; DdfStart
        dc.w         $94,$00d0         ; DdfStop


        dc.w         $102                ; BplCon1 - IL REGISTRO
        dc.b         $00                ; BplCon1 - IL BYTE NON UTILIZZATO!!!
MIOCON1:
        dc.b         $00                ; BplCon1 - IL BYTE UTILIZZATO!!!


        dc.w         $104,0                ; BplCon2
        dc.w         $108,0                ; Bpl1Mod
        dc.w         $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w         $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                        ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
```

```
        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

;        L'effetto nella copperlist: e' composto di un wait e un BPLCON1, i
;        wait aspettano una volta ogni 4 linee: $34,$38,$3c....
;        Nei $dff102 ci sono gia' i valori dell'"ONDA": 1,2,3,4...3,2,1.

        DC.W        $3007,$FFFE,$102
CON1EFFETTO:
        DC.W        $00
        DC.W        $3407,$FFFE,$102,$00
        DC.W        $3807,$FFFE,$102,$00
        DC.W        $3C07,$FFFE,$102,$11
        DC.W        $4007,$FFFE,$102,$11
        DC.W        $4407,$FFFE,$102,$11
        DC.W        $4807,$FFFE,$102,$11
        DC.W        $4C07,$FFFE,$102,$22
        DC.W        $5007,$FFFE,$102,$22
        DC.W        $5407,$FFFE,$102,$22
        DC.W        $5807,$FFFE,$102,$33
        DC.W        $5C07,$FFFE,$102,$33
        DC.W        $6007,$FFFE,$102,$44
        DC.W        $6407,$FFFE,$102,$44
        DC.W        $6807,$FFFE,$102,$55
        DC.W        $6C07,$FFFE,$102,$66
        DC.W        $7007,$FFFE,$102,$77
        DC.W        $7407,$FFFE,$102,$88
        DC.W        $7807,$FFFE,$102,$88
        DC.W        $7C07,$FFFE,$102,$99
        DC.W        $8007,$FFFE,$102,$99
        DC.W        $8407,$FFFE,$102,$aa
        DC.W        $8807,$FFFE,$102,$aa
        DC.W        $8C07,$FFFE,$102,$aa
        DC.W        $9007,$FFFE,$102,$99
        DC.W        $9407,$FFFE,$102,$99
        DC.W        $9807,$FFFE,$102,$88
        DC.W        $9C07,$FFFE,$102,$88
        DC.W        $A007,$FFFE,$102,$77
        DC.W        $A407,$FFFE,$102,$66
        DC.W        $A807,$FFFE,$102,$55
        DC.W        $AC07,$FFFE,$102,$44
        DC.W        $B007,$FFFE,$102,$44
        DC.W        $B407,$FFFE,$102,$33
        DC.W        $B807,$FFFE,$102,$33
        DC.W        $BC07,$FFFE,$102,$22
        DC.W        $C007,$FFFE,$102,$22
        DC.W        $C407,$FFFE,$102,$22
        DC.W        $C807,$FFFE,$102,$11
        DC.W        $CC07,$FFFE,$102,$11
        DC.W        $D007,$FFFE,$102,$11
        DC.W        $D407,$FFFE,$102,$11
        DC.W        $D807,$FFFE,$102,$00
        DC.W        $DC07,$FFFE,$102,$00
        DC.W        $E007,$FFFE,$102,$00
        DC.W        $E407,$FFFE,$102
ULTIMOVALORE:
```

```
        DC.W        $00

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                       ; convertita col KEFCON, fatta di
                                       ; 3 bitplanes consecutivi

        end
```

Questo effetto di ondulazione e' un classico nell'Amiga. Per risparmiare qua non ondula ogni linea separatamente ma ogni quattro linee, ma almeno ha una routine con un loop veloce per scorrere i valori del $102 presenti nella copperlist.

La routine in questa lezione puo' essere utilizzata per "ruotare" qualsiasi gruppo di word, dunque puo' servire anche per effetti di scorrimento dei colori, o qualsiasi altro effetto.

21.9 Lezione5i

```
; Lezione5i.s        SCORRIMENTO DI UNA FIGURA IN ALTO E IN BASSO DI TUTTA LA CHIP
;                MEMORY USANDO PUNTATORI AI PITPLANES NELLA COPPERLIST
;                TASTO SINISTRO PER SPOSTARSI IN AVANTI, DESTRO PER SPOSTARSI
;                INDIETRO, ENTRAMBI PER USCIRE.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;        Nota: qua i bitplane li lasciamo puntare a $000000, ossia
;        all' inizio della CHIP MEMORY

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti
Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!

        btst        #2,$dff016        ; se il tasto destro e' premuto
        bne.s        NonGiu                ; scorri giu!, oppure vai a NonGiu

        bsr.s        VaiGiu                ; tasto destro premuto, scorri giu!
```

```
Nongiu:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        beq.s        Scorrisu        ; se si, scorri in su
        bra.s        mouse                ; no? allora ripeti il ciclo il prossimo FRAME

Scorrisu:
        bsr.w        VaiSu                ; fa scorrere la figura in alto

        btst        #2,$dff016        ; se anche il tasto destro e' premuto allora
        bne.s        mouse                ; sono premuti entrambi, esci, oppure "MOUSE"

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0


;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)

VAIGIU:
        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0        ; in d0 - il contrario della routine che
        sub.l        #80*3,d0        ; sottraiamo 80*3, ossia 3 linee, facendo
                                ; scorrere in BASSO la figura
        bra.s        Finito


VAISU:
        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0        ; in d0 - il contrario della routine che
        add.l        #80*3,d0        ; Aggiungiamo 80*3, ossia 3 linee, facendo
                                ; scorrere in ALTO la figura
        bra.w        finito


Finito:                                ; PUNTIAMO I PUNTATORI BITPLANES
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        rts
```

```
        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$003c        ; DdfStart HIRES normale
        dc.w        $94,$00d4        ; DdfStop HIRES normale
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%1001001000000000        ; bits 12/15 accesi!! 1 bitplane
                                      ; hires 640x256, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$2ae        ; color1

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

        end
```

Con questo programmino potete vedere il contenuto della vostra CHIP RAM, infatti viene visualizzato 1 bitplane in hires, che punta all'indirizzo $00000 ossia all'inizio della CHIP RAM dell'Amiga. Premendo il tasto sinistro del mouse potete incrementare l'indirizzo visualizzato, scorrendo tutta la memoria, in cui noterete lo schermo del wordbench, quello dell'ASMONE, nonche' evetuali figure rimaste in memoria, ad esempio se avete giocato ad un gioco prima di eseguire questo listato probabilmente troverete gli sfondi e i personaggi del gioco sempre in memoria, in quanto la memoria non viene cancellata al reset, ma soltanto spegnendo il cumputer. Col tasto destro potete indietreggiare per centrare un'immagine che vi interessa; per uscire dovete premere entrambi i bottoni. provate a fare un'esperimento caricando vari videogiochi, resettando ed eseguendo questo programmino, per rintracciare in memoria quello che e' rimasto.
Se volete velocizzare lo scorrimento dovete aumentare il valore aggiunto ai bitplane, basta che sia un multiplo di 80 (infatti per scorrere di una linea in HIRES, essendo largo 640 bit per linea anziche' 320, occorre il doppio di 40, che abbiamo fin qua usato per gli schermi in LOWRES).
Nel listato lo schermo scorre di 3 linee alla volta:

```
        sub.l        #80*3,d0        ; sottraiamo 80*3, ossia 3 linee
```

Per farlo scorrere col TURBO provate con 80*10 o maggiori.
Se per curiosita' volete sapere a che indirizzo sta un bitplane che vedete sullo schermo, uscite in quel punto e scrivete "M BPLPOINTERS":

```
XXXXXX 00 E0 00 02 00 E2 10 C0 ... (00 e0 = bplpointerH, 00 e2 l'altro BPLP)

ossia $00E0,$0002,$00E2,$10C0 ......

In questo esempio l'indirizzo e' $0002 10c0, ossia $210c0
```

21.10 Lezione5l

```
; Lezione5l.s        Effetto "ALLUNGAMENTO" fatto alternando moduli normali e -40

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;        Puntiamo i bitplanes in copperlist

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

        SECTION        GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

; COPPERLIST CHE "ALLUNGA"

        dc.l        $8907fffe               ; wait linea $89
        dc.w        $108,-40,$10a,-40       ; modulo -40, ripetizione ultima linea
        dc.l        $9007fffe               ; aspetto 7 linee -saranno tutte uguali
        dc.w        $108,0,$10a,0               ; poi faccio avanzare di una linea
        dc.l        $9107fffe               ; e la linea seguente...
        dc.w        $108,-40,$10a,-40       ; rimetto il modulo a FLOOD
        dc.l        $9807fffe               ; aspetto 7 linee -saranno tutte uguali
        dc.w        $108,0,$10a,0               ; faccio avanzare alla linea dopo
        dc.l        $9907fffe               ; poi...
        dc.w        $108,-40,$10a,-40       ; mi ripeto la linea per 7 linee col
        dc.l        $a007fffe               ; modulo a -40
        dc.w        $108,0,$10a,0               ; avanzo di una linea... ECCETERA.
        dc.l        $a107fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $a807fffe
        dc.w        $108,0,$10a,0
        dc.l        $a907fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $b007fffe
        dc.w        $108,0,$10a,0
        dc.l        $b107fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $b807fffe
        dc.w        $108,0,$10a,0
        dc.l        $b907fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $c007fffe
        dc.w        $108,0,$10a,0
```

```
        dc.l        $c107fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $c807fffe
        dc.w        $108,0,$10a,0
        dc.l        $c907fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $d007fffe
        dc.w        $108,0,$10a,0
        dc.l        $d107fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $d807fffe
        dc.w        $108,0,$10a,0
        dc.l        $d907fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $e007fffe
        dc.w        $108,0,$10a,0
        dc.l        $e107fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $e807fffe
        dc.w        $108,0,$10a,0
        dc.l        $e907fffe
        dc.w        $108,-40,$10a,-40
        dc.l        $f007fffe
        dc.w        $108,0,$10a,0        ; ritorno alla normalita'

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                       ; convertita col KEFCON, fatta di
                                       ; 3 bitplanes consecutivi

        end
```

Questo e' uno degli altri utilizzi dell'effetto "FLOOD" fatto con i moduli, infatti e' piuttosto facile "allungare" una figura o simulare dei pixel piu' lunghi del normale alternando dei moduli -40, che allungano, a dei moduli normalmente a zero, che fanno scattare la linea seguente, la quale sara' poi allungata facendola seguire da un altro modulo -40 mantenuto per qualche linea. In questo esempio l'allungamento e' un *8, infatti la linea viene fatta avanzare solo una volta ogni 8 pixel, infatti i moduli -40 vengono distanziati con i wait di 7 linee, e tra questi allungamenti sono poste delle linee a modulo normale, che dunque fanno scattare alla linea successiva terminata la visualizzazione, ma la linea seguente c'e' subito un'altro modulo negativo che fa ripetere la nuova linea per 7 righe, piu' quella con modulo normale che fa scattare nuovamente la linea nuova quando va "a capo".
Cambiando la distanza tra i wait si possono creare interessanti effetti di ondulazione in stile "zoom".

21.11 Lezione5m

```
; Lezione5m.s        SPOSTAMENTO DELLA FINESTRA VIDEO CON IL DIWSTART ($dff08e)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
```

```
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0               ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst        #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s        Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w        SuGiuDIW        ; scorre in alto e in basso col DIWSTART

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta                ; Se si, non andare avanti, aspetta!

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Questa routine agisce semplicemente sul byte YY del $dff08e in
;        copperlist, il DIWSTART; questo registro definisce l'inizio della
;        finestra video, che puo' essere "centrata", come si puo' fare
```

```
;        dalle preferences del workBench. Nel nostro caso semplicemente
;        facciamo "cominciare" la finestra video piu' in basso, per cui
;        si sposta quello che contiene. In questo caso a differenza dello
;        scroll che abbiamo visto con i bitplanes, non viene visualizzato
;        nulla "sopra" la figura, perche' spostiamo proprio la "finestra", e
;        al di fuori di essa non vengono visualizzati i bitplane.
;        Un'aspetto interessante della routine puo' essere il fatto che viene
;        usata una word etichettata come COUNTER per aspettare 35 fotogrammi
;        prima di agire, per creare un ritardo quando il logo e' in alto
;        prima di scendere; ho usato anche due istruzioni "nuove", che non
;        avevamo ancora visto, ma che sono utilissime in questa routine; si
;        tratta del BHI, che e' un'istruzione della famiglia BEQ/BNE, che salta
;        alla routine se il risultato del CMP, ossia del COMPARA, e' che
;        il valore e' SUPERIORE, in questo caso BHI.s LOGOD fa saltare a LOGOD
;        solo quando il COUNTER ha raggiunto il valore 35, nonche' le volte
;        dopo, in cui sara' a 36,37 eccetera, comunque SUPERIORE a 35.
;        L'altra istruzione e' il BCHG, che significa BIT CHANGE, ossia
;        "scambia il bit", e' della famiglia del BTST, e "scambia" il bit
;        indicato, ossia: un BCHG #1,label agisce sul bit 1 di quella label
;        facendolo diventare 1 se era 0, 0 se era 1.

SuGiuDIW:
        ADDQ.W        #1,COUNTER        ; segnamo l'esecuzione
        CMPI.W        #35,COUNTER        ; sono passato almeno 35 fotogrammi?
        BHI.S        LOGOD                ; se si esegui la routine
        RTS                        ; altrimenti torna senza eseguirla

LOGOD:
        BTST        #1,FLAGDIW        ; Dobbiamo andare in alto?
        BEQ.S        UP                ; Se si eseguiamo la routine "UP"
        SUBQ.B        #2,DIWSCX        ; Vai in alto a passi di 2, piu' velocemente
        CMPI.B        #$2c,DIWSCX        ; Siamo in cima? (valore normale $2c81)
        BEQ.S        CHANGEUPDOWN2        ; se si cambiamo la direzione di scroll
        RTS

UP:
        ADDQ.B        #1,DIWSCX        ; Vai in basso a passi di 1, lentamente
        CMPI.B        #$70,DIWSCX        ; Siamo in fondo? (posizione $70)
        BEQ.S        CHANGEUPDOWN        ; se si, cambiamo direzione di scorrimento
        RTS

CHANGEUPDOWN
        BCHG        #1,FLAGDIW        ; scambiamo il bit della direzione
        RTS

CHANGEUPDOWN2
        BCHG        #1,FLAGDIW        ; scambiamo il bit della direzione
        CLR.W        COUNTER                ; e azzeriamo il COUNTER, siamo al termine!
        RTS

FLAGDIW:
        dc.w        0

COUNTER:
        dc.w        0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
```

```
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E
DIWSCX:
        dc.w        $2c81        ; DIWSTRT = $YYXX Inizio finestra video

        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                    ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                    ; convertita col KEFCON, fatta di
                                    ; 3 bitplanes consecutivi

        end
```

Lezione5m2

```
; Lezione5m2.s        "CHIUSURA" DELLA FINESTRA VIDEO CON I DIWSTART/STOP ($8e/$90)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES
```

```
        MOVE.L      #PIC,d0              ; in d0 mettiamo l'indirizzo della PIC,
        LEA       BPLPOINTERS,A1       ; puntatori nella COPPERLIST
        MOVEQ       #2,D1               ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w      d0,6(a1)           ; copia la word BASSA dell'indirizzo del plane
        swap       d0                  ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w      d0,2(a1)           ; copia la word ALTA dell'indirizzo del plane
        swap       d0                  ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L       #40*256,d0         ; + lunghezza bitplane -> prossimo bitplane
        addq.w      #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra       d1,POINTBP          ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l      #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w      d0,$dff088               ; Facciamo partire la COP
        move.w      #0,$dff1fc               ; Disattiva l'AGA
        move.w      #$c00,$dff106               ; Disattiva l'AGA

mouse:
        cmpi.b      #$ff,$dff006        ; Siamo alla linea 255?
        bne.s       mouse               ; Se non ancora, non andare avanti

        btst       #2,$dff016          ; se il tasto destro e' premuto salta
        beq.s       Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w       DIWORIZZONTALE        ; mostra la funzione dei DIWSTART e DIWSTOP

Aspetta:
        cmpi.b      #$ff,$dff006        ; Siamo alla linea 255?
        beq.s       Aspetta               ; Se si, non andare avanti, aspetta!

        btst       #6,$bfe001          ; tasto sinistro del mouse premuto?
        bne.s       mouse                ; se no, torna a mouse:

        move.l      OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w      d0,$dff088                ; facciamo partire la vecchia cop

        move.l      4.w,a6
        jsr        -$7e(a6)         ; Enable - riabilita il Multitasking
        move.l      gfxbase(PC),a1         ; Base della libreria da chiudere
        jsr        -$19e(a6)         ; Closelibrary - chiudo la graphics lib
        rts                         ; USCITA DAL PROGRAMMA

;       Dati

GfxName:
        dc.b       "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l       0        ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l       0

; Questa routine porta a $ff DIWXSTART incrementandolo di uno ogni volta e
; a $00 DIWXSTOP decrementandolo di uno ogni volta. Quando entrambi i valori
; sono raggiunti la routine esce senza modificare niente

DIWORIZZONTALE:
        CMPI.B      #$FF,DIWXSTART        ; Siamo arrivati al massimo DIWSTART?
        BEQ.S       FINITO               ; se si, non possiamo procedere oltre
        ADDQ.B      #1,DIWXSTART        ; se no, allora aggiungiamo 1
FINITO:
```

```
        TST.B        DIWXSTOP        ; Siamo arrivati al minimo DIWSTOP? ($00)
        BEQ.S        FINITO2               ; se si non possiamo calare oltre
        SUBQ.B        #1,DIWXSTOP        ; se no, allora sottraiamo1
FINITO2:
        RTS                        ; Uscita dalla routine


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E                ; DIWSTART - Inizio finestra video
        dc.b        $2c                ; DIWSTRT $YY
DIWXSTART:
        dc.b        $81                ; DIWSTRT $XX (lo incrementiamo fino a $ff)

        dc.w        $90                ; DIWSTOP - Fine finestra video
        dc.b        $2c                ; DiwStop YY
DIWXSTOP:
        dc.b        $c1                ; DiwStop XX (lo caliamo fino a $00)
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $ca07,$fffe
        dc.w        $180,$456        ; nota: il colore di fondo non viene coinvolto
                            ; dal diwstart-diwstop

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                ; convertita col KEFCON, fatta di
                                ; 3 bitplanes consecutivi

        end
```

```
Questo listato mostra come si possa diminuire la grandezza della finestra
video in senso orizzontale: se per esempio visualizzassimo solo delle figure
al centro dello schermo, potremmo "risparmiare" lavoro al copper, dunque
guadagnare velocita' per altri lavori, semplicemente restringendo la finestra
facendoci entrare la figura ed escludendo i "vuoti" laterali, oppure si possono
fare effetti di "chiusura" dello schermo. Avrete notato pero' che non si puo'
chiudere del tutto "lo schermo", ma rimane una linea, e che questa linea non
e' al centro dello schermo, ma spostata verso destra. Infatti il limite che
si puo; raggiungere nel "RESTRINGIMENTO" del visualizzabile e' proprio a quella
linea, infatti e' la posizione DIWSTART XX = $FF e DIWSTOP XX = $00.
Avrete notato anche che questi registri influiscono sui bitplanes, e non sul
colore di sfondo!
```

Lezione5m3

```
; Lezione5m3.s        "CHIUSURA" DELLA FINESTRA VIDEO CON I DIWSTART/STOP ($8e/$90)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr          -$78(a6)        ; Disable - ferma il multitasking
        lea          GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr          -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA          BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ          #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        btst         #2,$dff016        ; se il tasto destro e' premuto salta
        beq.s         Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w         DIWVERTICALE        ; mostra la funzione dei DIWSTART e DIWSTOP

Aspetta:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        beq.s         Aspetta                ; Se si, non andare avanti, aspetta!
```

```
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

; Questa routine porta a $95 DIWYSTART incrementandolo di uno ogni volta e
; a $95 DIWYSTOP decrementandolo di uno ogni volta. Quando entrambi i valori
; sono raggiunti la routine esce senza modificare niente
; Da notare che il DIWSTIO qua parte da $fe e non da $ff+$2c come al solito.

DIWVERTICALE:
        CMPI.B        #$95,DIWYSTOP        ; Siamo arrivati al DIWSTOP giusto?
        BEQ.S        FINITO                ; se si, non dobbiamo procedere oltre
        ADDQ.B        #1,DIWYSTART        ; aggiungiamo 1 allo start
        SUBQ.B        #1,DIWYSTOP        ; sottraiamo 1 allo stop
FINITO:
        RTS                        ; Uscita dalla routine


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E                ; DIWSTART - Inizio finestra video
DIWYSTART:
        dc.b        $2c                ; DIWSTRT $YY
        dc.b        $81                ; DIWSTRT $XX (lo incrementiamo fino a $ff)

        dc.w        $90                ; DIWSTOP - Fine finestra video
DIWYSTOP:
        dc.b        $fe                ; DiwStop YY (partiamo dalla linea $fe!!)
        dc.b        $c1                ; DiwStop XX (lo caliamo fino a $00)
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
```

```
                ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                        ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi

        end
```

Questo listato mostra come si possa diminuire la grandezza della finestra
video in senso verticale: se per esempio visualizzassimo solo delle figure
nella parte alta dello schermo, potremmo restringere la finestra, "tagliando"
la parte sotto una certa linea; il byte YY del diwstart/stop e' uguale a quello
del WAIT: un wait $2c07,$fffe aspetta la prima linea bitplane visualizzata,
infatti il DIWSTART e' $2c81. Giunto alla linea $FF, il DIWSTOP riparte da
ZERO: dunque aspettando col diwstop la posizione $2cc1, aspetta la linea
$ff+$2c, ossia 299, ma le linee effettivamente usate per visualizzare i
bitplane sono 256: dalla $2c (44) alla 299.
In questo esempio influisce anche il fatto che la figura si sposta in basso
assieme all'inizio della visualizzazione della finestra video.
Per vedere meglio cosa succede sul video, sostituite 3 bitplanes "pieni" ossia
con tutti i bit ad 1:

```
PIC:
        dcb.b        40*256*3,$FF
```

Lezione5m4

```
; Lezione5m4.s        "CHIUSURA" DELLA FINESTRA VIDEO CON I DIWSTART/STOP ($8e/$90)

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia
```

```
;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA         BPLPOINTERS,A1         ; puntatori nella COPPERLIST
        MOVEQ         #2,D1                 ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)         ; copia la word BASSA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)         ; copia la word ALTA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                 ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP         ; Rifai D1 volte POINTBP (D1=num of bitplanes)
;
        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b        #$ff,$dff006         ; Siamo alla linea 255?
        bne.s         mouse                ; Se non ancora, non andare avanti

        btst         #2,$dff016         ; se il tasto destro e' premuto salta
        beq.s         Aspetta                ; la routine dello scroll, bloccandolo

        bsr.w         DIWORIZZONTALE         ; mostrano la funzione dei DIWSTART e DIWSTOP
        bsr.w         DIWVERTICALE         ;

Aspetta:
        cmpi.b        #$ff,$dff006         ; Siamo alla linea 255?
        beq.s         Aspetta                ; Se si, non andare avanti, aspetta!

        btst         #6,$bfe001         ; tasto sinistro del mouse premuto?
        bne.s         mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080         ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)         ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1         ; Base della libreria da chiudere
        jsr         -$19e(a6)         ; Closelibrary - chiudo la graphics lib
        rts                         ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l         0         ; della graphics.library

OldCop:                          ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l         0


DIWORIZZONTALE:
        CMPI.B        #$FF,DIWXSTART         ; Siamo arrivati al massimo DIWSTART?
        BEQ.S         FINITO                ; se si, non possiamo procedere oltre
        ADDQ.B        #2,DIWXSTART         ; se no, allora aggiungiamo 1
FINITO:
```

```
        TST.B       DIWXSTOP        ; Siamo arrivati al minimo DIWSTOP? ($00)
        BEQ.S       FINITO2             ; se si non possiamo calare oltre
        SUBQ.B       #2,DIWXSTOP        ; se no, allora sottraiamo1
FINITO2:
        RTS                         ; Uscita dalla routine


DIWVERTICALE:
        CMPI.B       #$95,DIWYSTOP        ; Siamo arrivati al DIWSTOP giusto?
        BEQ.S       FINITO3               ; se si, non dobbiamo procedere oltre
        ADDQ.B       #1,DIWYSTART        ; aggiungiamo 1 allo start
        SUBQ.B       #1,DIWYSTOP        ; sottraiamo 1 allo stop
FINITO3:
        RTS                         ; Uscita dalla routine

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E                 ; DIWSTART - Inizio finestra video
DIWYSTART:
        dc.b        $2c                 ; DIWSTRT $YY
DIWXSTART:
        dc.b        $81                 ; DIWSTRT $XX (lo incrementiamo fino a $ff)

        dc.w        $90                 ; DIWSTOP - Fine finestra video
DIWYSTOP:
        dc.b        $fe                 ; DiwStop YY
DIWXSTOP:
        dc.b        $c1                 ; DiwStop XX (lo caliamo fino a $00)
        dc.w        $92,$0038         ; DdfStart
        dc.w        $94,$00d0         ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)
                                    ; 3 bitplanes lowres, non lace
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000         ; color0
        dc.w        $0182,$475         ; color1
        dc.w        $0184,$fff         ; color2
        dc.w        $0186,$ccc         ; color3
        dc.w        $0188,$999         ; color4
        dc.w        $018a,$232         ; color5
        dc.w        $018c,$777         ; color6
        dc.w        $018e,$444         ; color7

        dc.w        $ca07,$fffe
        dc.w        $180,$456         ; nota: il colore di fondo non viene coinvolto
                                ; dal diwstart-diwstop
```

```
        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        figura

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                       ; convertita col KEFCON, fatta di
                                       ; 3 bitplanes consecutivi

        end
```

In questo listato sono modificate sia le XX che le YY dei DIWSTART e DIWSTOP
per strangolare la figura.

21.12 Lezione5n

```
; Lezione5n.s        FUSIONE DI 3 EFFETTI COPPER + FIGURA AD 8 COLORI con
;                EFFETTI $dff102 e bitplane pointers

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANES

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP

        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        mt_init                ; Inizializza routine musicale

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        bsr.w        muovicopper        ; barra rossa sotto linea $ff
        bsr.s        CopperDestSin        ; Routine di scorrimento destra/sinistra
        BSR.w        scrollcolors        ; scorrimento ciclico dei colori
        bsr.w        ScorriPlanes        ; scorrimento su-giu della figura
```

```
        bsr.w       Ondula              ; Ondulazione tramite molti $dff102
        bsr.w       mt_music        ; Suona la musica

Aspetta:
        cmpi.b       #$ff,$dff006       ; Siamo alla linea 255?
        beq.s       Aspetta              ; Se si, non andare avanti, aspetta la linea
                            ; seguente, altrimenti MuoviCopper viene
                            ; rieseguito

        btst       #6,$bfe001       ; tasto sinistro del mouse premuto?
        bne.s       mouse               ; se no, torna a mouse:

        bsr.w       mt_end              ; Termina la routine musicale

        move.l       OldCop(PC),$dff080       ; Puntiamo la cop di sistema
        move.w       d0,$dff088               ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l       gfxbase(PC),a1       ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA


;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

; ***************************************************************************
; *                BARRA A SCORRIMENTO ORIZZONTALE (Lezione3h.s)                    *
; ***************************************************************************

CopperDESTSIN:
        CMPI.W       #85,DestraFlag               ; VAIDESTRA eseguita 85 volte?
        BNE.S       VAIDESTRA               ; se non ancora, rieseguila
        CMPI.W       #85,SinistraFlag       ; VAISINISTRA eseguita 85 volte?
        BNE.S       VAISINISTRA               ; se non ancora, rieseguila
        CLR.W       DestraFlag        ; la routine VAISINISTRA e' stata eseguita
        CLR.W       SinistraFlag        ; 85 volte, riparti
        RTS                        ; TORNIAMO AL LOOP mouse


VAIDESTRA:                        ; questa routine sposta la barra verso DESTRA
        lea        CopBar+1,A0        ; Mettiamo in A0 l'indirizzo del primo XX
        move.w       #29-1,D2        ; dobbiamo cambiare 29 wait (usiamo un DBRA)
DestraLoop:
        addq.b       #2,(a0)               ; aggiungiamo 2 alla coordinata X del wait
        ADD.W       #16,a0               ; andiamo al prossimo wait da cambiare
        dbra       D2,DestraLoop        ; ciclo eseguito d2 volte
        addq.w       #1,DestraFlag        ; segnamo che abbiamo eseguito VAIDESTRA
        RTS                        ; TORNIAMO AL LOOP mouse


VAISINISTRA:                        ; questa routine sposta la barra verso SINISTRA
        lea        CopBar+1,A0
```

```
        move.w        #29-1,D2        ; dobbiamo cambiare 29 wait
SinistraLoop:
        subq.b        #2,(a0)               ; sottraiamo 2 alla coordinata X del wait
        ADD.W        #16,a0               ; andiamo al prossimo wait da cambiare
        dbra        D2,SinistraLoop        ; ciclo eseguito d2 volte
        addq.w        #1,SinistraFlag ; Annotiamo lo spostamento
        RTS                        ; TORNIAMO AL LOOP mouse


DestraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w        0        ; che e' stata eseguita VAIDESTRA

SinistraFlag:                ; In questa word viene tenuto il conto delle volte
        dc.w    0        ; che e' stata eseguita VAISINISTRA

; ***************************************************************************
; *                BARRA ROSSA SOTTO LA LINEA $FF (Lezione3f.s)                  *
; ***************************************************************************

MuoviCopper:
        LEA        BARRA,a0
        TST.B        SuGiu                ; Dobbiamo salire o scendere?
        beq.w        VAIGIU
        cmpi.b        #$0a,(a0)        ; siamo arrivati alla linea $0a+$ff? (265)
        beq.s        MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        subq.b        #1,(a0)
        subq.b        #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        subq.b        #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        subq.b        #1,8*3(a0)
        subq.b        #1,8*4(a0)
        subq.b        #1,8*5(a0)
        subq.b        #1,8*6(a0)
        subq.b        #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        subq.b        #1,8*8(a0)        ; barra rossa ogni volta per farla salire!
        subq.b        #1,8*9(a0)
        rts

MettiGiu:
        clr.b        SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                        ; fara' saltare alla routine VAIGIU, e
                                ; la barra scedera'

VAIGIU:
        cmpi.b        #$2c,8*9(a0)        ; siamo arrivati alla linea $2c?
        beq.s        MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        addq.b        #1,(a0)
        addq.b        #1,8(a0)        ; ora cambiamo gli altri wait: la distanza
        addq.b        #1,8*2(a0)        ; tra un wait e l'altro e' di 8 bytes
        addq.b        #1,8*3(a0)
        addq.b        #1,8*4(a0)
        addq.b        #1,8*5(a0)
        addq.b        #1,8*6(a0)
        addq.b        #1,8*7(a0)        ; qua dobbiamo modificare tutti i 9 wait della
        addq.b        #1,8*8(a0)        ; barra rossa ogni volta per farla scendere!
        addq.b        #1,8*9(a0)
        rts

MettiSu:
        move.b        #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                        ; significa che dobbiamo risalire.
```

```
SuGiu:
        dc.b        0,0

; ***************************************************************************
; *                SCORRIMENTO CICLICO DEI COLORI (Lezione3E.s)                  *
; ***************************************************************************

Scrollcolors:
        move.w        col2,col1        ; col2 copiato in col1
        move.w        col3,col2        ; col3 copiato in col2
        move.w        col4,col3        ; col4 copiato in col3
        move.w        col5,col4        ; col5 copiato in col4
        move.w        col6,col5        ; col6 copiato in col5
        move.w        col7,col6        ; col7 copiato in col6
        move.w        col8,col7        ; col8 copiato in col7
        move.w        col9,col8        ; col9 copiato in col8
        move.w        col10,col9        ; col10 copiato in col9
        move.w        col11,col10        ; col11 copiato in col10
        move.w        col12,col11        ; col12 copiato in col11
        move.w        col13,col12        ; col13 copiato in col12
        move.w        col14,col13        ; col14 copiato in col13
        move.w        col1,col14        ; col1 copiato in col14
        rts


; ***************************************************************************
; *        SCORRIMENTO IN ALTO E IN BASSO DELLA FIGURA (da Lezione5g.s)          *
; ***************************************************************************

;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)

ScorriPlanes:
        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0        ; in d0 - il contrario della routine che
                                ; punta i bitplanes! Qua invece di mettere
                                ; l'indirizzo lo prendiamo!!!

        TST.B        SuGiu3                ; Dobbiamo salire o scendere? se SuGiu e'
                                ; azzerata, (cioe' il TST verifica il BEQ)
                                ; allora saltiamo a VAIGIU, se invece e' a $FF
                                ; (se cioe' questo TST non e' verificato)
                                ; continuiamo salendo (facendo dei sub)
        beq.w        VAIGIU3
        cmp.l        #PIC-(40*18),d0        ; siamo arrivati abbastanza in BASSO?
        beq.w        MettiGiu3        ; se si, siamo in fondo e dobbiamo risalire
        sub.l        #40,d0                ; sottraiamo 40, ossia 1 linea, facendo
                                ; scorrere in BASSO la figura
        bra.s        Finito3

MettiGiu3:
        clr.b        SuGiu3                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        bra.s        Finito3                ; fara' saltare alla routine VAIGIU

VAIGIU3:
        cmpi.l        #PIC+(40*130),d0 ; siamo arrivati abbastanza in ALTO?
        beq.s        MettiSu3        ; se si, siamo in fondo e dobbiamo risalire
        add.l        #40,d0                ; Aggiungiamo 40, ossia 1 linea, facendo
                                ; scorrere in ALTO la figura
        bra.s        finito3
```

```
MettiSu3:
        move.b        #$ff,SuGiu3        ; Quando la label SuGiu non e' a zero,
        rts                         ; significa che dobbiamo risalire.

Finito3:                        ; PUNTIAMO I PUNTATORI BITPLANES
        LEA         BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ         #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w        d0,6(a1)         ; copia la word BASSA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)         ; copia la word ALTA dell'indirizzo del plane
        swap         d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP2        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu3:
        dc.b         0,0


; ***************************************************************************
; *        EFFETTO DI ONDULAZIONE TRAMITE MOLTI $dff102 (Lezione5h.s)          *
; ***************************************************************************


Ondula:
        LEA         CON1EFFETTO+8,A0 ; Indirizzo word sorgente in a0
        LEA         CON1EFFETTO,A1        ; Indirizzo delle word destinazione in a1
        MOVEQ         #19,D2                ; 20 bplcon1 da cambiare in COPLIST
SCAMBIA:
        MOVE.W         (A0),(A1)        ; copia due word consecutive - scorrimento!
        ADDQ.W         #8,A0                ; prossima coppia di word
        ADDQ.W         #8,A1                ; prossima coppia di word
        DBRA         D2,SCAMBIA        ; ripeti "SCAMBIA" il numero giusto di VOLTE

        MOVE.W         CON1EFFETTO,ULTIMOVALORE ; per rendere infinito il ciclo
        RTS                                ; copiamo il primo valore nell'ultimo
                                        ; ogni volta.

; ***************************************************************************
; *                ROUTINE CHE SUONA MUSICHE SOUNDTRACKER/PROTRACKER          *
; ***************************************************************************

        include         "music.s"        ; routine 100% funzionante su tutti gli Amiga

; ***************************************************************************
; *                                SUPER COPPERLIST                          *
; ***************************************************************************

        SECTION         GRAPHIC,DATA_C

COPPERLIST:

        ; Facciamo puntare gli sprite a ZERO, per eliminarli, o ce li troviamo
        ; in giro impazziti a disturbare!!!

        dc.w         $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000
```

```
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

; Per uno schermo a 3 bitplanes: (8 colori)

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)

;        Facciamo puntare i bitplanes direttamente mettendo nella copperlist
;        i registri $dff0e0 e seguenti qua di seguito con gli indirizzi
;        dei bitplanes che saranno messi dalla routine POINTBP

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

;        Gli 8 colori della figura qua sono resi piu' "verdi"

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$070        ; color1
        dc.w        $0184,$0f0        ; color2
        dc.w        $0186,$0c0        ; color3
        dc.w        $0188,$090        ; color4
        dc.w        $018a,$030        ; color5
        dc.w        $018c,$070        ; color6
        dc.w        $018e,$040        ; color7

;        L'effetto di Lezione3e.s spostato piu' in ALTO

        dc.w        $2c07,$fffe        ; aspettiamo la linea 154 ($9a in esadecimale)
        dc.w        $180                ; REGISTRO COLOR0
col1:
        dc.w        $0f0                ; VALORE DEL COLOR 0 (che sara' modificato)
        dc.w        $2d07,$fffe ; aspettiamo la linea 155 (non sara' modificata)
        dc.w        $180                ; REGISTRO COLOR0 (non sara' modificato)
col2:
        dc.w        $0d0                ; VALORE DEL COLOR 0 (sara' modificato)
        dc.w        $2e07,$fffe        ; aspettiamo la linea 156 (non modificato,ecc.)
        dc.w        $180                ; REGISTRO COLOR0
col3:
        dc.w        $0b0                ; VALORE DEL COLOR 0
        dc.w         $2f07,$fffe        ; aspettiamo la linea 157
        dc.w        $180                ; REGISTRO COLOR0
col4:
        dc.w        $090                ; VALORE DEL COLOR 0
        dc.w        $3007,$fffe        ; aspettiamo la linea 158
        dc.w        $180                ; REGISTRO COLOR0
col5:
        dc.w        $070                ; VALORE DEL COLOR 0
        dc.w        $3107,$fffe        ; aspettiamo la linea 159
        dc.w        $180                ; REGISTRO COLOR0
```

```
col6:
        dc.w        $050                ; VALORE DEL COLOR 0
        dc.w        $3207,$fffe         ; aspettiamo la linea 160
        dc.w        $180                ; REGISTRO COLOR0
col7:
        dc.w        $030                ; VALORE DEL COLOR 0
        dc.w        $3307,$fffe         ; aspettiamo la linea 161
        dc.w        $180                ; color0... (ora avete capito i commenti,
col8:                                   ; posso anche smettere di metterli da qua!)
        dc.w        $030
        dc.w        $3407,$fffe         ; linea 162
        dc.w        $180
col9:
        dc.w        $050
        dc.w        $3507,$fffe         ;  linea 163
        dc.w        $180
col10:
        dc.w        $070
        dc.w        $3607,$fffe         ;  linea 164
        dc.w        $180
col11:
        dc.w        $090
        dc.w        $3707,$fffe         ;  linea 165
        dc.w        $180
col12:
        dc.w        $0b0
        dc.w        $3807,$fffe         ;  linea 166
        dc.w        $180
col13:
        dc.w        $0d0
        dc.w        $3907,$fffe         ;  linea 167
        dc.w        $180
col14:
        dc.w        $0f0
        dc.w         $3a07,$fffe        ;  linea 168

        dc.w        $0180,$000         ; color0        ; colori reali della figura
        dc.w        $0182,$475         ; color1
        dc.w        $0184,$fff         ; color2
        dc.w        $0186,$ccc         ; color3
        dc.w        $0188,$999         ; color4
        dc.w        $018a,$232         ; color5
        dc.w        $018c,$777         ; color6
        dc.w        $018e,$444         ; color7

;        Effetto copper dell'ondulazione col $dff102 di Lezione5h.s "ristretto"

        DC.W        $102
CON1EFFETTO:
        dc.w        $000
        DC.W        $4007,$FFFE,$102,$00
        DC.W        $4407,$FFFE,$102,$11
        DC.W        $4807,$FFFE,$102,$11
        DC.W        $4C07,$FFFE,$102,$22
        DC.W        $5007,$FFFE,$102,$33
        DC.W        $5407,$FFFE,$102,$44
        DC.W        $5807,$FFFE,$102,$66
        DC.W        $5C07,$FFFE,$102,$66
        DC.W        $6007,$FFFE,$102,$77
        DC.W        $6407,$FFFE,$102,$77
        DC.W        $6807,$FFFE,$102,$77
        DC.W        $6C07,$FFFE,$102,$66
```

```
        DC.W        $7007,$FFFE,$102,$66
        DC.W        $7407,$FFFE,$102,$55
        DC.W        $7807,$FFFE,$102,$33
        DC.W        $7C07,$FFFE,$102,$22
        DC.W        $8007,$FFFE,$102,$11
        DC.W        $8407,$FFFE,$102,$11
        DC.W        $8807,$FFFE,$102,$00
        DC.W        $8C07,$FFFE,$102
ULTIMOVALORE:
        DC.W        $00

;        EFFETTO DELLA LEZIONE3h.s

        dc.w        $9007,$fffe        ; aspettiamo l'inizio della linea
        dc.w        $180,$000        ; grigio al minimo, ossia NERO!!!
CopBar:
        dc.w        $9031,$fffe        ; wait che cambiamo ($9033,$9035,$9037...)
        dc.w        $180,$100        ; colore rosso
        dc.w        $9107,$fffe        ; wait che non cambiamo (Inizio linea)
        dc.w        $180,$111        ; colore GRIGIO (parte dall'inizio linea fino
        dc.w        $9131,$fffe        ; a questo WAIT, che noi cambiaremo...
        dc.w        $180,$200        ; dopo il quale comincia il ROSSO

;            WAIT FISSI (poi grigio) - WAIT DA CAMBIARE (seguiti dal rosso)

        dc.w        $9207,$fffe,$180,$120,$9231,$fffe,$180,$301 ; linea 3
        dc.w        $9307,$fffe,$180,$230,$9331,$fffe,$180,$401 ; linea 4
        dc.w        $9407,$fffe,$180,$240,$9431,$fffe,$180,$502 ; linea 5
        dc.w        $9507,$fffe,$180,$350,$9531,$fffe,$180,$603 ; ....
        dc.w        $9607,$fffe,$180,$360,$9631,$fffe,$180,$703
        dc.w        $9707,$fffe,$180,$470,$9731,$fffe,$180,$803
        dc.w        $9807,$fffe,$180,$580,$9831,$fffe,$180,$904
        dc.w        $9907,$fffe,$180,$690,$9931,$fffe,$180,$a04
        dc.w        $9a07,$fffe,$180,$7a0,$9a31,$fffe,$180,$b04
        dc.w        $9b07,$fffe,$180,$8b0,$9b31,$fffe,$180,$c05
        dc.w        $9c07,$fffe,$180,$9c0,$9c31,$fffe,$180,$d05
        dc.w        $9d07,$fffe,$180,$ad0,$9d31,$fffe,$180,$e05
        dc.w        $9e07,$fffe,$180,$be0,$9e31,$fffe,$180,$f05
        dc.w        $9f07,$fffe,$180,$cf0,$9f31,$fffe,$180,$e05
        dc.w        $a007,$fffe,$180,$be0,$a031,$fffe,$180,$d05
        dc.w        $a107,$fffe,$180,$ad0,$a131,$fffe,$180,$c05
        dc.w        $a207,$fffe,$180,$9c0,$a231,$fffe,$180,$b04
        dc.w        $a307,$fffe,$180,$8b0,$a331,$fffe,$180,$a04
        dc.w        $a407,$fffe,$180,$7a0,$a431,$fffe,$180,$904
        dc.w        $a507,$fffe,$180,$690,$a531,$fffe,$180,$803
        dc.w        $a607,$fffe,$180,$580,$a631,$fffe,$180,$703
        dc.w        $a707,$fffe,$180,$470,$a731,$fffe,$180,$603
        dc.w        $a807,$fffe,$180,$360,$a831,$fffe,$180,$502
        dc.w        $a907,$fffe,$180,$250,$a931,$fffe,$180,$402
        dc.w        $aa07,$fffe,$180,$140,$aa31,$fffe,$180,$301
        dc.w        $ab07,$fffe,$180,$130,$ab31,$fffe,$180,$202
        dc.w        $ac07,$fffe,$180,$120,$ac31,$fffe,$180,$103
        dc.w        $ad07,$fffe,$180,$111,$ad31,$fffe,$180,$004

        dc.w        $ae07,$fffe
        dc.w        $180,$002
        dc.w        $af07,$fffe
        dc.w        $180,$003

;        Effetto specchio "cilindrico" della Lezione3g.s (+ridefinizione colori)

        dc.w        $0182,$235        ; color1
```

```
        dc.w    $0184,$99e      ; color2
        dc.w    $0186,$88c      ; color3
        dc.w    $0188,$659      ; color4
        dc.w    $018a,$122      ; color5
        dc.w    $018c,$337      ; color6
        dc.w    $018e,$224      ; color7

        dc.w    $b007,$fffe
        dc.w    $180,$004       ; Color0
        dc.w    $102,$011       ; bplcon1
        dc.w    $108,-40*7      ; Bpl1Mod - specchio dimezzato 5 volte
        dc.w    $10a,-40*7      ; Bpl2Mod
        dc.w    $b307,$fffe

        dc.w    $180,$006       ; Color0
        dc.w    $102,$022       ; bplcon1
        dc.w    $108,-40*6      ; Bpl1Mod - specchio dimezzato 4 volte
        dc.w    $10a,-40*6      ; Bpl2Mod

        dc.w    $b607,$fffe

        dc.w    $0182,$245      ; color1
        dc.w    $0184,$9cf      ; color2
        dc.w    $0186,$89c      ; color3
        dc.w    $0188,$669      ; color4
        dc.w    $018a,$132      ; color5
        dc.w    $018c,$347      ; color6
        dc.w    $018e,$234      ; color7

        dc.w    $180,$008       ; Color0
        dc.w    $102,$033       ; bplcon1
        dc.w    $108,-40*5      ; Bpl1Mod - specchio dimezzato 3 volte
        dc.w    $10a,-40*5      ; Bpl2Mod

        dc.w    $bb07,$fffe

        dc.w    $180,$00a       ; Color0
        dc.w    $102,$044       ; bplcon1
        dc.w    $108,-40*4      ; Bpl1Mod - specchio dimezzato 2 volte
        dc.w    $10a,-40*4      ; Bpl2Mod

        dc.w    $c307,$fffe

        dc.w    $0182,$355      ; color1
        dc.w    $0184,$abf      ; color2
        dc.w    $0186,$9ac      ; color3
        dc.w    $0188,$779      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$457      ; color6
        dc.w    $018e,$344      ; color7
        dc.w    $180,$00c       ; Color0
        dc.w    $102,$055       ; bplcon1
        dc.w    $108,-40*3      ; Bpl1Mod - specchio dimezzato
        dc.w    $10a,-40*3      ; Bpl2Mod

        dc.w    $d007,$fffe

        dc.w    $180,$00e       ; Color0
        dc.w    $102,$066       ; bplcon1
        dc.w    $108,-40*2      ; Bpl1Mod - specchio normale
        dc.w    $10a,-40*2      ; Bpl2Mod
```

```
        dc.w    $d607,$fffe
        dc.w    $0182,$465      ; color1
        dc.w    $0184,$cdf      ; color2
        dc.w    $0186,$bbc      ; color3
        dc.w    $0188,$889      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$557      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $180,$00f       ; Color0
        dc.w    $102,$077       ; bplcon1
        dc.w    $108,-40        ; Bpl1Mod - FLOOD, linee ripetute per
        dc.w    $10a,-40        ; Bpl2Mod - effetto centrale di ingrandimento

        dc.w    $da07,$fffe

        dc.w    $0182,$355      ; color1
        dc.w    $0184,$abf      ; color2
        dc.w    $0186,$9ac      ; color3
        dc.w    $0188,$779      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$457      ; color6
        dc.w    $018e,$344      ; color7
        dc.w    $180,$00e       ; Color0
        dc.w    $102,$066       ; bplcon1
        dc.w    $108,-40*2      ; Bpl1Mod - specchio normale
        dc.w    $10a,-40*2      ; Bpl2Mod

        dc.w    $e007,$fffe

        dc.w    $0182,$245      ; color1
        dc.w    $0184,$9cf      ; color2
        dc.w    $0186,$89c      ; color3
        dc.w    $0188,$669      ; color4
        dc.w    $018a,$132      ; color5
        dc.w    $018c,$347      ; color6
        dc.w    $018e,$234      ; color7
        dc.w    $180,$00c       ; Color0
        dc.w    $102,$055       ; bplcon1
        dc.w    $108,-40*3      ; Bpl1Mod - specchio dimezzato
        dc.w    $10a,-40*3      ; Bpl2Mod

        dc.w    $ed07,$fffe

        dc.w    $180,$00a       ; Color0
        dc.w    $102,$044       ; bplcon1
        dc.w    $108,-40*4      ; Bpl1Mod - specchio dimezzato 2 volte
        dc.w    $10a,-40*4      ; Bpl2Mod

        dc.w    $f507,$fffe

        dc.w    $0182,$235      ; color1
        dc.w    $0184,$99e      ; color2
        dc.w    $0186,$88c      ; color3
        dc.w    $0188,$659      ; color4
        dc.w    $018a,$122      ; color5
        dc.w    $018c,$337      ; color6
        dc.w    $018e,$224      ; color7
        dc.w    $180,$008       ; Color0
        dc.w    $102,$033       ; bplcon1
        dc.w    $108,-40*5      ; Bpl1Mod - specchio dimezzato 3 volte
        dc.w    $10a,-40*5      ; Bpl2Mod
```

```
        dc.w        $fa07,$fffe

        dc.w        $180,$006       ; Color0
        dc.w        $102,$022       ; bplcon1
        dc.w        $108,-40*6       ; Bpl1Mod - specchio dimezzato 4 volte
        dc.w        $10a,-40*6       ; Bpl2Mod

        dc.w        $fd07,$fffe

        dc.w        $180,$004       ; Color0
        dc.w        $102,$011       ; bplcon1
        dc.w        $108,-40*7       ; Bpl1Mod - specchio dimezzato 5 volte
        dc.w        $10a,-40*7       ; Bpl2Mod

        dc.w        $ff07,$fffe

        dc.w        $180,$002       ; Color0
        dc.w        $102,$000       ; bplcon1
        dc.w        $108,-40       ; ferma l'immagine per evitare di visualizzare
        dc.w        $10a,-40       ; i byte prima della RAW

;        Effetto della lezione3f.s

        dc.w        $ffdf,$fffe       ; ATTENZIONE! WAIT ALLA FINE LINEA $FF!
                            ; i wait dopo questo sono sotto la linea
                            ; $FF e ripartono da $00!!

        dc.w        $0107,$FFFE       ; una barretta fissa verde SOTTO la linea $FF!
        dc.w        $180,$010
        dc.w        $0207,$FFFE
        dc.w        $180,$020
        dc.w        $0307,$FFFE
        dc.w        $180,$030
        dc.w        $0407,$FFFE
        dc.w        $180,$040
        dc.w        $0507,$FFFE
        dc.w        $180,$030
        dc.w        $0607,$FFFE
        dc.w        $180,$020
        dc.w        $0707,$FFFE
        dc.w        $180,$010
        dc.w        $0807,$FFFE
        dc.w        $180,$000

BARRA:
        dc.w        $0907,$FFFE       ; aspetto la linea $79
        dc.w        $180,$300       ; inizio la barra rossa: rosso a 3
        dc.w        $0a07,$FFFE       ; linea seguente
        dc.w        $180,$600       ; rosso a 6
        dc.w        $0b07,$FFFE
        dc.w        $180,$900       ; rosso a 9
        dc.w        $0c07,$FFFE
        dc.w        $180,$c00       ; rosso a 12
        dc.w        $0d07,$FFFE
        dc.w        $180,$f00       ; rosso a 15 (al massimo)
        dc.w        $0e07,$FFFE
        dc.w        $180,$c00       ; rosso a 12
        dc.w        $0f07,$FFFE
        dc.w        $180,$900       ; rosso a 9
        dc.w        $1007,$FFFE
        dc.w        $180,$600       ; rosso a 6
```

```
        dc.w        $1107,$FFFE
        dc.w        $180,$300        ; rosso a 3
        dc.w        $1207,$FFFE
        dc.w        $180,$000        ; colore NERO

        dc.w        $FFFF,$FFFE        ; FINE DELLA COPPERLIST


; ***************************************************************************
; *                        FIGURA AD 8 COLORI 320x256                       *
; ***************************************************************************

        dcb.b       40*98,0               ; spazio azzerato

PIC:
        incbin       "amiga.320*256*3"        ; qua carichiamo la figura in RAW,
                                        ; convertita col KEFCON, fatta di
                                        ; 3 bitplanes consecutivi
        dcb.b       40*8,0                ; spazio azzerato

; ***************************************************************************
; *                                 MUSICA PROTRACKER                       *
; ***************************************************************************

mt_data:
        incbin       "mod.purple-shades"

        end

; ***************************************************************************
```

Questo listato non e' altro che Lezione4c.s a cui ho aggiunto Lezione4g.s
e Lezione4h.s, le uniche modifiche sono due:
1) Ho dovuto diminuire l'effetto di "ondulato" come numero di WAIT, per farlo
entrare tra un'effetto e l'altro, passando da 45 a 20.
2) Ho cambiato la palette della figura nella parte superiore, rendendola verde
come se la figura si "infilasse" dentro l'effetto "scrollColors", e ho
cambiato i colori qua e la' per migliorare (e allungare) la SUPERCOPPERLIST!

La vera notita' e' l'inserimento della routine che suona la musica!
Intanto per cominciare anziche' inserirla all'interno del listato ho preferito
utilizzare la direttiva dell'ASMONE "INCLUDE", che mi permette, appunto, di
INCLUDERE un pezzo di listato nel mio listato.
Vediamo dunque come si fa a fornire di musica le nostre produzioni: come prima
cosa bisogna chiarire che la musica e' in un formato particolare, in questo
caso PROTRACKER, non si tratta di un pezzo "CAMPIONATO" col digitalizzatore e
risuonato. Ci sono vari programmi per comporre musiche, il piu' usato e' il
protracker (compatibile soundtracker e noisetracker), che salva la musica
nel formato MOD, infatti spesso le musiche in questo formato cominciano per
MOD. Non e' detto pero' che si debba usare sempre una musica protracker, certi
giochi o demo Amiga, specialmente quelli piu' vecchi, hanno musiche composte
con programmi come MED, OCTAMED, FUTURE COMPOSER, SOUNDMONITOR, OKTALYZER, ma
in tal caso bisogna far "suonare" la musica con la routine addetta a suonare
tali formati musicali. Infatti assieme al programma musicale solitamente c'e'
la routine di REPLAY, che puo' essere inclusa nel listato per risuonarla.
Oggigiorno il 99% delle produzioni Amiga usano musiche Protracker, o comunque
sottospecie del protracker, ossia routines che compattano o ottimizzano un
modulo in formato protracker e lo fanno diventare "prorunner" o "propacker",
dunque in questo corso ho incluso la routine che suona musiche PROTRACKER,
compatibile con moduli NOISETRACKER e SOUNDTRACKER vari, che tra l'altro ho
modificato per renderla compatibile al 100% con i microprocessori 68020+ anche
con le CACHE attive, infatti originariamente questa replay routine aveva dei

problemi con processori troppo veloci che causavano il "taglio" e la "perdita" di alcune note durante l'esecuzione. Dunque la routine "music.s" suona bene anche sull'Amiga 4000.
Per utilizzarla basta inserirla nel listato, o col comando "I", oppure potete caricarla in un'altro buffer di testo e copiarla nel vostro listato. Personalmente preferisco risparmiare spazio nei listati e la includo con la direttiva "INCLUDE", che in pratica fa assemblare la routine come se fosse stata inserita manualmente, ma si risparmiano i 21k della sua lunghezza: immaginate di avere 5 sorgenti, ed in ognuno volete mettere la musica:

```
        sorgente1.s        12234 bytes
        sorgente2.s        23523 bytes
        sorgente3.s        29382 bytes
        sorgente4.s        78343 bytes
        sorgente5.s        10482 bytes
        sorgente6.s        14925 bytes
        sorgente7.s        29482 bytes
```

Insieme sono lunghi circa 200k, mentre dopo aver aggiunto a tutti i 21k della REPLAY-ROUTINE occuperebbero complessivamente circa 300k! Mentre aggiungendo solo la linea

```
        include        "music.s"
```

L'aumento sarebbe di pochi bytes, e il risultato lo stesso.
L'unico particolare e' che, come nell'INCBIN, bisogna trovarsi nella directory dove si trova il file da includere, o bisogna scrivere tutto il percorso:

```
        include        "df0:sorgenti2/music.s"
```

Una volta all'interno del listato, tramite INCLUDE o inserimento, la routine va fatta funzionare. FACILISSIMO! Basta eseguire "mt_init" prima del loop MOUSE per inizializzarla, eseguire un "mt_music" ogni FOTOGRAMMA per suonare, ed eseguire mt_end" alla fine prima di uscire per terminare e chiudere i canali audio:

```
        bsr.w        mt_init                ; Inizializza routine musicale

mouse:
        cmpi.b        #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse                ; Se non ancora, non andare avanti

        bsr.w        MiaRoutineGrafica
        bsr.w        mt_music

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        bsr.w        mt_end                ; Termina la routine musicale
```

La musica ovviamente deve essere caricata, basta caricarla con INCBIN alla label "mt_data":

```
mt_data:
        incbin        "mod.purple-shades"
```

La musica presente nel disco del corso e' di HI-LITE dei VISION FACTORY, una musica di qualche annetto fa, la ho scelta anche perche' e' lunga solo 13k!
Se volete far suonare una vostra musica basta caricarla con l'INCBIN:

```
mt_data:
```

```
        incbin        "df1:modules/mod.MIAMUSICA"        ; ad esempio!
```

CAPITOLO 22

LEZIONE 6

22.1 Lezione6a

```
; Lezione6a.s        STAMPIAMO UNA DEI CARATTERI SULLO SCHERMO!!!

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa una parola sullo schermo

mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
```

```
        move.w        d0,$dff088               ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr         -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                         ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel

TESTO:
        dc.b        'A'        ; il testo da stampare. Qua solo una "A", ossia $41

        EVEN        ; allinea a indirizzo pari


PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0),D2                ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        RTS


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000
```

```
        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE


        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "nice.fnt"        ; senza caratteri ALT

        SECTION        MIOPLANE,BSS_C        ; Le SECTION BSS devono essere fatte di
                                ; soli ZERI!!! si usa il DS.b per definire
                                ; quanti zeri contenga la section.

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Una "A" e' apparsa sul nostro monitor!!! Nell'angolo in alto a sinistra.
Potete cambiare parola da stampare, ma non e' una grande modifica stampare
una "B" anziche' una "A".

* MODIFICA 1:

Provate a far stampare solo meta' carattere, ossia le sue prime 4 linee:

```
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
;        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
;        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
;        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
;        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "
```

Ogni linea e' un byte, ossia 8 BIT

```
        12345678

        ...###.. linea        1 - 8 bit, 1 byte
        ..#...#. 2
        ..#...#. 3
        ..#####. 4
        ..#...#. 5
```

```
        ..#...#. 6
        ..#...#. 7
        ........ 8

* MODIFICA 2:

Provate a togliere l'EVEN dalla stringa:

        dc.b        "A"

Assemblando l'ASMONE vi comunichera' l'errore: "Word at ODD address", ossia
"INDIRIZZO DISPARI!!". Bastera' rimettere lo zero a posto o aggiungere EVEN.


* MODIFICA 3:

Per cambiare la posizione della "A" basta cambiare la destinazione del PRINT:

PRINT:
        LEA         TESTO(PC),A0
        LEA         BITPLANE+(40*120),A3 ; Indirizzo destinazione

In questo modo stampiamo 120 linee piu' in basso, al centro dello schermo.
Per far avanzare il carattere basta aggiungere dei bytes:

        LEA         BITPLANE+19+(40*120),A3 ; Indirizzo destinazione

In questo modo lo facciamo avanzare di 19 bytes, e viene stampato al ventesimo
byte, la meta' dello schermo (che e' di 40 bytes).

* MODIFICA 4:

Proviamo a visualizzare il carattere in un bitplane in HIRES: Per fare cio'
eseguite queste modifiche:

Nella routine, essendo lo schermo hires largo 80 byte per linea anziche' 40:

        MOVE.B      (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B      (A2)+,80(A3)        ; stampa LA LINEA 2  " "
        MOVE.B      (A2)+,80*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B      (A2)+,80*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B      (A2)+,80*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B      (A2)+,80*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B      (A2)+,80*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B      (A2)+,80*7(A3)        ; stampa LA LINEA 8  " "

Nella copperlist: settare il BIT 15 in BPLCON0, attivando l'HIRES

                ; 5432109876543210
        dc.w    $100,%1001001000000000        ; 1 bitplane HIRES 640x256

E modificare il DDFSTART/DDFSTOP per lo schermo HIRES, pena il "TAGLIO" delle
prime linee a sinistra. Se non modificate questi due registri infatti la "A"
non viene visualizzata se e' al bordo sinistro.

        dc.w        $92,$003c        ; DdfStart HIRES normale
        dc.w        $94,$00d4        ; DdfStop HIRES normale

Infine nella SECTION BSS: dobbiamo ingrandire il BITPLANE!

        ds.b        80*256        ; un bitplane hires 640x256
```

22.2 Lezione6b

```
; Lezione6a.s        STAMPIAMO UNA RIGA DI TESTO SULLO SCHERMO!!!

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa una riga di testo sullo schermo

mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel

PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #40-1,D0        ; NUMERO COLONNE PER RIGA: 40 (ossia il numero
                            ; di caratteri presenti in una riga).
```

```
PRINTCHAR2:
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA

        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A3+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        RTS


                ; numero caratteri per linea: 40
TESTO:                ;                1111111111222222222233333333334
             ;         12345678901234567890123456789012345678901234567890
        dc.b        ' PRIMA RIGA  Sullo Schermo! 123 prova   '

        EVEN



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
```

```
        dc.w        $0182,$19a        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
;        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
        incbin        "nice.fnt"        ; Carattere stretto

        SECTION        MIOPLANE,BSS_C        ; Le SECTION BSS devono essere fatte di
                                ; soli ZERI!!! si usa il DS.b per definire
                                ; quanti zeri contenga la section.

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

In una riga si possono scrivere molte cose. Per stampare 40 caratteri basta fare un ciclo DBRA, qua chiamato PRINTCHAR2:

```
        MOVEQ        #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
```

Essendo ogni carattere "largo" un byte, in una colonna ci possono essere 40 caratteri in LOWRES, o 80 in HIRES. Prima di rifare il ciclo per stampare il prossimo carattere c'e' un ADD che sposta al prossimo byte, ossia alla posizione seguente dove stampare il carattere adiacente:

```
        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA
```

Potete scegliere tra 3 font diversi, basta togliere e mettere i ; agli incbin per caricare quello che vi interessa.

Se volete visualizzare la linea in hires eseguite le modifiche come nella Lezione6a.s, in piu' potete "allungare" il testo fino ad 80 caratteri, in questo caso dovete modificare il loop PRINTCHAR in MOVEQ #80-1,d0. Il testo lo potete anche scrivere in due linee per ogni riga:

```
        dc.b        ' PRIMA RIGA  Sullo Schermo! 123 prova   ' 0-40
        dc.b        ' sono sempre sulla prima riga hires!!   ' 41-80

        dc.b        " SECONDA RIGA!........................."
        dc.b        "........................................"

        dc.b        " TERZA RIGA!...... eccetera.
```

22.3 Lezione6c

```
; Lezione6c.s        STAMPIAMO VARIE RIGHE DI TESTO SULLO SCHERMO!!!

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
```

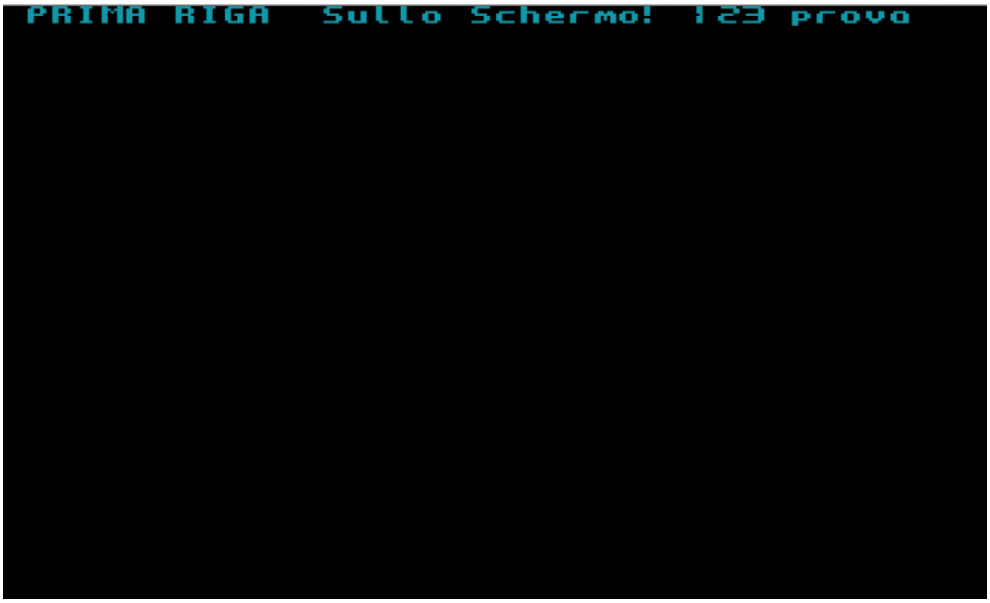


Figura 22.1: Lezione 6b

```
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l       d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l       d0,a6
        move.l       $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L       #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w       d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w       d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l       #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w       d0,$dff088                ; Facciamo partire la COP
        move.w       #0,$dff1fc                ; Disattiva l'AGA
        move.w       #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa le linee di testo sullo schermo

mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l       gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
```

```
        rts                         ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0         ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel

PRINT:
        LEA         TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA         BITPLANE,A3         ; Indirizzo del bitplane destinazione in a3
        MOVEQ         #23-1,D3        ; NUMERO RIGHE DA STAMPARE: 23
PRINTRIGA:
        MOVEQ         #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
        MOVEQ         #0,D2                ; Pulisci d2
        MOVE.B         (A0)+,D2         ; Prossimo carattere in d2
        SUB.B         #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W         #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L         D2,A2
        ADD.L         #FONT,A2         ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B         (A2)+,(A3)         ; stampa LA LINEA 1 del carattere
        MOVE.B         (A2)+,40(A3)         ; stampa LA LINEA 2  " "
        MOVE.B         (A2)+,40*2(A3)         ; stampa LA LINEA 3  " "
        MOVE.B         (A2)+,40*3(A3)         ; stampa LA LINEA 4  " "
        MOVE.B         (A2)+,40*4(A3)         ; stampa LA LINEA 5  " "
        MOVE.B         (A2)+,40*5(A3)         ; stampa LA LINEA 6  " "
        MOVE.B         (A2)+,40*6(A3)         ; stampa LA LINEA 7  " "
        MOVE.B         (A2)+,40*7(A3)         ; stampa LA LINEA 8  " "

        ADDQ.w         #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA         D0,PRINTCHAR2         ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W         #40*7,A3         ; ANDIAMO A CAPO

        DBRA         D3,PRINTRIGA         ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 40
TESTO:             ;                 1111111111222222222233333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b         '   PRIMA RIGA                           ' ; 1
        dc.b         '                SECONDA RIGA            ' ; 2
        dc.b         '     /\  /                              ' ; 3
        dc.b         '    /  \/                               ' ; 4
        dc.b         '                                        ' ; 5
```

```
        dc.b        '        SESTA RIGA                    ' ; 6
        dc.b        '                                      ' ; 7
        dc.b        '                                      ' ; 8
        dc.b        'FABIO CIUCCI COMMUNICATION INTERNATIONAL' ; 9
        dc.b        '                                      ' ; 10
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}    ' ; 11
        dc.b        '                                      ' ; 12
        dc.b        '     LA PALINGENETICA OBLITERAZIONE   ' ; 15
        dc.b        '                                      ' ; 25
        dc.b        '                                      ' ; 16
        dc.b        '  Nel mezzo del cammin di nostra vita ' ; 17
        dc.b        '                                      ' ; 18
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa  ' ; 19
        dc.b        '                                      ' ; 20
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA   ' ; 21
        dc.b        '                                      ' ; 22
        dc.b        '  AHI Quanto a DIR QUAL ERA...        ' ; 23
        dc.b        '                                      ' ; 24
        dc.b        '                                      ' ; 25
        dc.b        '                                      ' ; 26
        dc.b        '                                      ' ; 27

        EVEN



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE

        dc.w        $6c07,$fffe        ; Sfumatura alla linea di testo 9
        dc.w        $182,$451        ; linea1 del carattere
        dc.w        $6d07,$fffe
        dc.w        $182,$671        ; linea 2
        dc.w        $6e07,$fffe
        dc.w        $182,$891        ; linea 3
        dc.w        $6f07,$fffe
        dc.w        $182,$ab1        ; linea 4
        dc.w        $7007,$fffe
        dc.w        $182,$781        ; linea 5
        dc.w        $7107,$fffe
```

```
        dc.w        $182,$561        ; linea 6
        dc.w        $7207,$fffe
        dc.w        $182,$451        ; linea 7  l'ultima perche' la 8 e' azzerata
                                ;           per fare spaziatura tra le linee

        dc.w        $7307,$fffe
        dc.w        $182,$19a        ; colore normale

        dc.w        $8c07,$fffe        ; Sfumatura alla linea di testo 11
        dc.w        $182,$516        ; linea1 del carattere
        dc.w        $8d07,$fffe
        dc.w        $182,$739        ; linea 2
        dc.w        $8e07,$fffe
        dc.w        $182,$95b        ; linea 3
        dc.w        $8f07,$fffe
        dc.w        $182,$c6f        ; linea 4
        dc.w        $9007,$fffe
        dc.w        $182,$84a        ; linea 5
        dc.w        $9107,$fffe
        dc.w        $182,$739        ; linea 6
        dc.w        $9207,$fffe
        dc.w        $182,$517        ; linea 7  l'ultima perche' la 8 e' azzerata


        dc.w        $9307,$fffe
        dc.w        $182,$19a        ; colore normale

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
;        incbin        "nice.fnt"        ; Carattere stretto

        SECTION        MIOPLANE,BSS_C        ; Le SECTION BSS devono essere fatte di
                                ; soli ZERI!!! si usa il DS.b per definire
                                ; quanti zeri contenga la section.

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Come avrete visto, il fatto che il font sia ad un solo colore non vieta di
cambiare il colore ogni linea con i WAIT del copper!
Per scrivere molte righe l'una sotto l'altra e' bastato "ANDARE A CAPO" e
stampare la linea seguente, per un numero di volte specificato in D3

```
        ADD.W        #40*7,A3        ; ANDIAMO A CAPO
        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE
```

NOTA: per andare a capo serve scendere di 7 linee. Per RIGA intendo RIGA DI
TESTO, alta 8 pixel, per linea intendo LINEA VIDEO effettiva.

Ecco perche' per andare a capo serve un "ADD.W #40*7,A3" :

Il problema puo' nascere dall'impressione di trovarsi gia' con l'indirizzo in
a3 all'ultima linea del carattere appena stampata, per cui verrebbe da pensare
che basti scattare avanti di 1 per trovarsi alla riga di testo successiva, ma

```
in realta' in A3 c'e' sempre e solo l'indirizzo della prima linea dei caratteri
infatti le atre 7 linee sono stampate tramite OFFSET:

        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

Ma il registro A3 punta sempre alla prima linea. Infatti ogni volta che viene
stampato un carattere, si avanza al carattere successivo aggiungendo 8 bit,
ossia 1 byte, all'indirizzo in A3, il quale poi puntera' alla prima linea del
carattere successivo:

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

A questo punto per stampare quel "carattere successivo" bastera' rieseguire la
routine con le distanze di indirizzamento (OFFSET).
Vediamo cosa succede quando ci troviamo ad aver stampato l'ultimo carattere a
destra, ossia l'ultimo di una riga: in A3 abbiamo l'indirizzo della prima linea
dell'ultimo carattere in questione, dopo le istruzioni che stampano facendo
l'offset da A3, c'e' l'istruzione che fa scattare A3 agli 8 bit successivi, in
questo caso fa scattare all'inizio della seconda linea. E' per questo che per
portare A3 alla prima linea della riga successiva basta scendere di 7 linee e
non di 8, perche' ci trovavamo gia' all'inizio della seconda linea.
```

Lezione6c2

```
; Lezione6c2.s        STAMPIAMO VARIE RIGHE DI TESTO SULLO SCHERMO!!!
;                - con font in binario MODIFICABILE FACILMENTE!!

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa le linee di testo sullo schermo
```

```
mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel

PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #25-1,D3        ; NUMERO RIGHE DA STAMPARE: 25
PRINTRIGA:
        MOVEQ        #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #40*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE
```

```
        RTS

;                       CARATTERI DISPONIBILI NEL FONT:
;
;          !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
;
;                CARATTERI CHE NON SONO NEL FONT, DA NON USARE:
;
;                   [\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
;
;
; NOTA: il carattere "@" stampa una faccia sorridente... perche' no?

                ; numero caratteri per linea: 40
TESTO:                                                          ;40 caratteri
        dc.b        "   PRIMA RIGA                          " ; 1
        dc.b        "               SECONDA RIGA            " ; 2
        dc.b        "  /  /                                 " ; 3
        dc.b        " /  /  NON POSSO STAMPARE L'ALTRA BARRA!" ; 4
        dc.b        "                                       " ; 5
        dc.b        "        SESTA RIGA                     " ; 6
        dc.b        "                                       " ; 7
        dc.b        "                                       " ; 8
        dc.b        "FABIO CIUCCI COMMUNICATION INTERNATIONAL" ; 9
        dc.b        "                                       " ; 10
        dc.b        "   ! #$%&'()*+,-./0123456789:;<=>? @   " ; 11
        dc.b        "                                       " ; 12
        dc.b        "     LA PALINGENETICA OBLITERAZIONE    " ; 15
        dc.b        "                                       " ; 25
        dc.b        "                                       " ; 16
        dc.b        "  NEL MEZZO DEL CAMMIN DI NOSTRA VITA  " ; 17
        dc.b        "                                       " ; 18
        dc.b        "    MI RITROVAI PER UNA SELVA OSCURA   " ; 19
        dc.b        "                                       " ; 20
        dc.b        "    CHE LA DIRITTA VIA ERA SMARRITA    " ; 21
        dc.b        "                                       " ; 22
        dc.b        "  AHI QUANTO A DIR QUAL ERA...         " ; 23
        dc.b        "                                       " ; 24
        dc.b        "       @ @ @ SOLO MAIUSCOLE @ @ @      " ; 25
        dc.b        "                                       " ; 26
        dc.b        "                                       " ; 27

        EVEN

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256
```

```
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$345        ; color0 - SFONDO
        dc.w        $0182,$bdf        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

;        caratteri:  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
;        ATTENZIONE! non ci sono: [\]^_‘abcdefghijklmnopqrstuvwxyz{|}~

; CONSIGLIO: Per scorrere in basso usate il cursore giu' + SHIFT e fate una
; pagina alla volta!!!

FONT:
; ' '
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; '!'
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00000000
; '"'
        dc.b        %00011011
        dc.b        %00011011
        dc.b        %00011011
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; '#'
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %01111111
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %00000000
; '$'
        dc.b        %00001000
        dc.b        %00011110
        dc.b        %00100000
        dc.b        %00011100
        dc.b        %00000010
        dc.b        %00111100
        dc.b        %00001000
```

```
        dc.b        %00000000
; '%'
        dc.b        %00000001
        dc.b        %00110011
        dc.b        %00110110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110110
        dc.b        %01100110
        dc.b        %00000000
; '&'
        dc.b        %00011000
        dc.b        %00100100
        dc.b        %00011000
        dc.b        %00011001
        dc.b        %00100110
        dc.b        %00111110
        dc.b        %00011001
        dc.b        %00000000
; "'"
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; "("
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00110000
        dc.b        %00110000
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00000000
; ")"
        dc.b        %00110000
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00000000
; "*"
        dc.b        %01100011
        dc.b        %00110110
        dc.b        %00011100
        dc.b        %01111111
        dc.b        %00011100
        dc.b        %00110110
        dc.b        %01100011
        dc.b        %00000000
; '+'
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %01111110
        dc.b        %00011000
        dc.b        %00011000
```

```
        dc.b        %00000000
        dc.b        %00000000
; ","
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00000000
; "-"
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %01111110
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; "."
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00000000
; "/"
        dc.b        %00000001
        dc.b        %00000011
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %01100000
        dc.b        %00000000
; '0'
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; '1'
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; '2'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %01100000
```

```
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000000
; '3'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00011111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; '4'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; '5'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; '6'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; '7'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; '8'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; '9'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111111
```

```
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; ':'
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
; ';'
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00000000
; "<"
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00000110
        dc.b        %00000000
; "="
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %01111110
        dc.b        %00000000
        dc.b        %01111110
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ">"
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00000110
        dc.b        %00000011
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00110000
        dc.b        %00000000
; '?'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00001111
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00000000
; "@"
        dc.b        %00000000       ; sorriso
        dc.b        %11100111
        dc.b        %11100111
```

```
        dc.b        %00000000
        dc.b        %00010000
        dc.b        %00011000
        dc.b        %10000001
        dc.b        %01111110
; "A"
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; "B"
        dc.b        %01111110
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111110
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111110
        dc.b        %00000000
; 'C'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000000
; 'D'
        dc.b        %01111110
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111110
        dc.b        %00000000
; 'E'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111100
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000000
; 'F'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111100
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %00000000
; 'G'
        dc.b        %01111111
        dc.b        %00000000
```

```
        dc.b        %01100000
        dc.b        %01100111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; 'H'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01101111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'I'
        dc.b        %00111111
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00111111
        dc.b        %00000000
; 'J'
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000000
; 'K'
        dc.b        %01100011
        dc.b        %01100110
        dc.b        %00001100
        dc.b        %01111000
        dc.b        %01101100
        dc.b        %01100110
        dc.b        %01100011
        dc.b        %00000000
; 'L'
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %00000000
        dc.b        %01111111
        dc.b        %00000000
; 'M'
        dc.b        %01100011
        dc.b        %01110111
        dc.b        %01101011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'N'
        dc.b        %01111111
```

```
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'O'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; 'P'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %00000000
; 'Q'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100111
        dc.b        %01111111
        dc.b        %00000000
; 'R'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111100
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'S'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; 'T'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; 'U'
```

```
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; 'V'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00110110
        dc.b        %00011100
        dc.b        %00000000
; 'W'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01101011
        dc.b        %01110111
        dc.b        %01100011
        dc.b        %00000000
; 'X'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00110110
        dc.b        %00001000
        dc.b        %00110110
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'Y'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; 'Z'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %01111111
        dc.b        %00000000
;
; mancano i caratteri in minuscolo... se avete la pazienza di disegnarli, fate
; pure! Oppure potete fare disegnini da comporre insieme...
;

        SECTION     MIOPLANE,BSS_C      ; Le SECTION BSS devono essere fatte di
                                        ; soli ZERI!!! si usa il DS.b per definire
                                        ; quanti zeri contenga la section.
```

```
BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Questo listato e' uguale a Lezione6c.s, ma il font e' "FATTO A MANO", infatti anziche' caricarlo e' nel listato in forma di dc.b in binario

```
                ;12345678
; "A"
        dc.b        %01111111        ;1
        dc.b        %00000011        ;2
        dc.b        %01100011        ;3
        dc.b        %01111111        ;4
        dc.b        %01100011        ;5
        dc.b        %01100011        ;6
        dc.b        %01100011        ;7
        dc.b        %00000000        ;8
```

Questa per esempio e' la "A". Attenzione a non usare caratteri minuscoli nel testo, perche' non sono nel font, in quanto chi lo ha fatto si deve essere stancato alla "Z" maiuscola. In realta' non c'erano nemmeno molti simboli come "*;<>=" e li ho aggiunti io. Ora apparira' piu' chiara anche come e' fatto il font! E intuirete che per fare un font di 16x16 dovete fare cosi':

```
                ;1234567890123456
; "A"
        dc.w        %0000111111111100        ;1
        dc.w        %0011111111111111        ;2
        dc.w        %0011110000001111        ;3
        dc.w        %0011110000001111        ;4
        dc.w        %0011110000001111        ;5
        dc.w        %0011110000001111        ;6
        dc.w        %0011111111111111        ;7
        dc.w        %0011111111111111        ;8
        dc.w        %0011110000001111        ;9
        dc.w        %0011110000001111        ;10
        dc.w        %0011110000001111        ;11
        dc.w        %0011110000001111        ;12
        dc.w        %0011110000001111        ;13
        dc.w        %0011110000001111        ;14
        dc.w        %0000000000000000        ;15
        dc.w        %0000000000000000        ;16
```

Ma conviene disegnarlo e convertirlo in RAW!

In questo listato vi consiglio di modificare il FONT, aggiungendo disegnini e simboli strani. Potreste farvi il FONT personale!

Lezione6c2b

```
; Lezione6c2.s        STAMPIAMO VARIE RIGHE DI TESTO SULLO SCHERMO!!!
;                - con font in binario MODIFICABILE FACILMENTE!!

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
```

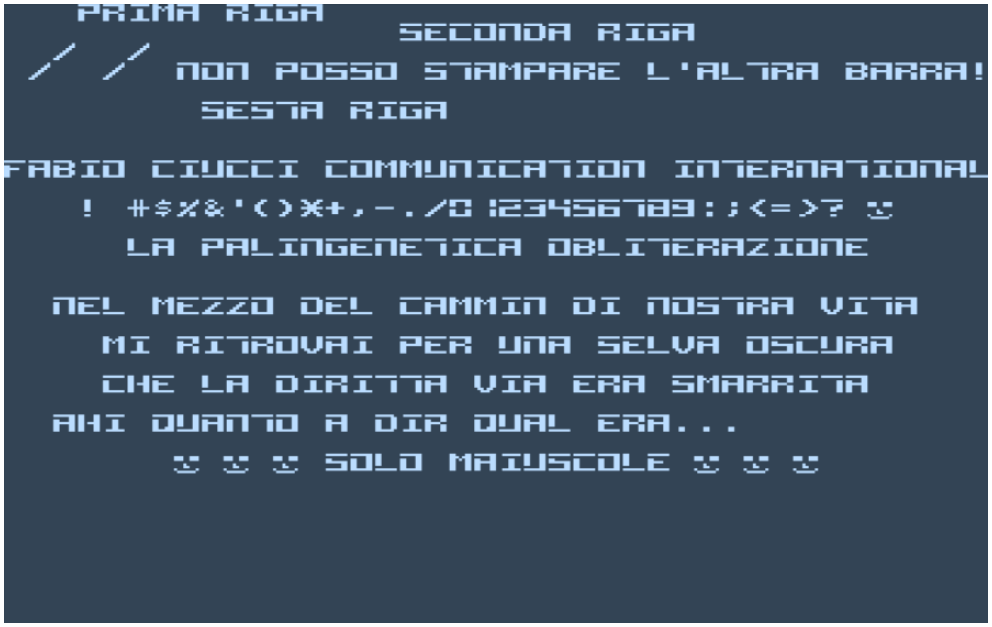


Figura 22.2: Lezione 6c2

```
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa le linee di testo sullo schermo

mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
```

```
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                         ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel

PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #25-1,D3        ; NUMERO RIGHE DA STAMPARE: 25
PRINTRIGA:
        MOVEQ        #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #40*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE

        RTS

;                        CARATTERI DISPONIBILI NEL FONT:
;
;          !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
;
;                CARATTERI CHE NON SONO NEL FONT, DA NON USARE:
;
;                   [\]^_`abcdefghijklmnopqrstuvwxyz{|}~
;
```

```
;
; NOTA: il carattere "@" stampa una faccia sorridente... perche' no?

                ; numero caratteri per linea: 40
TESTO:                                                          ;40 caratteri
        dc.b        "                                       " ; 1
        dc.b        "                                       " ; 2
        dc.b        "                                       " ; 3
        dc.b        "                                       " ; 4
        dc.b        "                                       " ; 5
        dc.b        "                                       " ; 6
        dc.b        "                                       " ; 7
        dc.b        "                                       " ; 8
        dc.b        "                                       " ; 9
        dc.b        "                                       " ; 10
        dc.b        "                                       " ; 11
        dc.b        "                                       " ; 12
        dc.b        "                                       " ; 15
        dc.b        "                                       " ; 25
        dc.b        "                                       " ; 16
        dc.b        "                                       " ; 17
        dc.b        "                                       " ; 18
        dc.b        "                                       " ; 19
        dc.b        "                                       " ; 20
        dc.b        "                                       " ; 21
        dc.b        "                                       " ; 22
        dc.b        "                                       " ; 23
        dc.b        "                                       " ; 24
        dc.b        "                                       " ; 25
        dc.b        "                                       " ; 26
        dc.b        "                                       " ; 27

        EVEN

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$345        ; color0 - SFONDO
        dc.w        $0182,$bdf        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8
```

```
;        caratteri:  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
;        ATTENZIONE! non ci sono: [\]^_‘abcdefghijklmnopqrstuvwxyz{|}~

; CONSIGLIO: Per scorrere in basso usate il cursore giu' + SHIFT e fate una
; pagina alla volta!!!

FONT:
; ' '
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; '!'
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00000000
; '"'
        dc.b        %00011011
        dc.b        %00011011
        dc.b        %00011011
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; '#'
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %01111111
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %00010100
        dc.b        %00000000
; '$'
        dc.b        %00001000
        dc.b        %00011110
        dc.b        %00100000
        dc.b        %00011100
        dc.b        %00000010
        dc.b        %00111100
        dc.b        %00001000
        dc.b        %00000000
; '%'
        dc.b        %00000001
        dc.b        %00110011
        dc.b        %00110110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110110
        dc.b        %01100110
        dc.b        %00000000
```

```
; '&'
        dc.b        %00011000
        dc.b        %00100100
        dc.b        %00011000
        dc.b        %00011001
        dc.b        %00100110
        dc.b        %00111110
        dc.b        %00011001
        dc.b        %00000000
; "'"
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; "("
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00110000
        dc.b        %00110000
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00000000
; ")"
        dc.b        %00110000
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00000000
; "*"
        dc.b        %01100011
        dc.b        %00110110
        dc.b        %00011100
        dc.b        %01111111
        dc.b        %00011100
        dc.b        %00110110
        dc.b        %01100011
        dc.b        %00000000
; '+'
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %01111110
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00000000
        dc.b        %00000000
; ","
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00110000
```

```
        dc.b        %00000000
; "-"
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %01111110
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; "."
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00011000
        dc.b        %00011000
        dc.b        %00000000
; "/"
        dc.b        %00000001
        dc.b        %00000011
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %01100000
        dc.b        %00000000
; '0'
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; '1'
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; '2'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000000
; '3'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00011111
        dc.b        %00000011
        dc.b        %00000011
```

```
        dc.b        %01111111
        dc.b        %00000000
; '4'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; '5'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; '6'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; '7'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000000
; '8'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000000
; '9'
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %00000000
; ':'
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00000000
```

```
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
; ';'
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00000000
; "<"
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00011000
        dc.b        %00110000
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00000110
        dc.b        %00000000
; "="
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %01111110
        dc.b        %00000000
        dc.b        %01111110
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ">"
        dc.b        %00011000
        dc.b        %00001100
        dc.b        %00000110
        dc.b        %00000011
        dc.b        %00000110
        dc.b        %00001100
        dc.b        %00110000
        dc.b        %00000000
; '?'
        dc.b        %01111111
        dc.b        %00000000
        dc.b        %00000011
        dc.b        %00001111
        dc.b        %00001100
        dc.b        %00000000
        dc.b        %00001100
        dc.b        %00000000
; "@"
        dc.b        %00000000       ; sorriso
        dc.b        %11100111
        dc.b        %11100111
        dc.b        %00000000
        dc.b        %00010000
        dc.b        %00011000
        dc.b        %10000001
        dc.b        %01111110
; "A"
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %01111111
        dc.b        %01111111
```

```
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
; "B"
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111110
        dc.b        %01111110
; 'C'
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
; 'D'
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
; 'E'
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111000
        dc.b        %01111000
; 'F'
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111000
        dc.b        %01111000
; 'G'
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01101111
        dc.b        %01101111
; 'H'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
```

```
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
; 'I'
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
; 'J'
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00000011
; 'K'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100111
        dc.b        %01100111
        dc.b        %01101110
        dc.b        %01101110
        dc.b        %01111100
        dc.b        %01111100
; 'L'
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
; 'M'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01110111
        dc.b        %01110111
        dc.b        %01110111
        dc.b        %01111111
        dc.b        %01101011
        dc.b        %00001000
; 'N'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01110011
        dc.b        %01110011
        dc.b        %01111011
        dc.b        %01111011
        dc.b        %01101111
        dc.b        %01101111
; 'O'
        dc.b        %00111110
        dc.b        %00111110
```

```
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
; 'P'
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
; 'Q'
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01101011
        dc.b        %01101011
; 'R'
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
; 'S'
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %00111110
        dc.b        %00111110
; 'T'
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
; 'U'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
; 'V'
        dc.b        %01100011
```

```
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01110111
        dc.b        %01110111
; 'W'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01101011
        dc.b        %01101011
; 'X'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01110111
        dc.b        %01110111
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %00011100
        dc.b        %00011100
; 'Y'
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00110111
        dc.b        %00110111
        dc.b        %00111110
        dc.b        %00111110
; 'Z'
        dc.b        %01111111       ;58
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00001110
        dc.b        %00001110
        dc.b        %00011100
        dc.b        %00011100
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
```

```
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; 'a'
        dc.b        %01111111       ;65
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'b'
        dc.b        %01111111       ;66
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111110
        dc.b        %01111110
; 'c'
        dc.b        %01100000       ;67
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %00000000
```

```
; 'd'
        dc.b        %01100011       ;68
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %00000000
; 'e'
        dc.b        %01111000       ;69
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00000000
; 'f'
        dc.b        %01111000       ;70
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %00000000
; 'g'
        dc.b        %01101111       ;71
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %00000000
; 'h'
        dc.b        %01111111       ;72
        dc.b        %01111111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'i'
        dc.b        %00001100       ;73
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00111111
        dc.b        %00111111
        dc.b        %00000000
; 'j'
        dc.b        %00000011       ;74
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %00110011
        dc.b        %00110011
        dc.b        %00011110
        dc.b        %00011110
```

```
        dc.b        %00000000
; 'k'
        dc.b        %01111100       ;75
        dc.b        %01111100
        dc.b        %01111100
        dc.b        %01101110
        dc.b        %01101110
        dc.b        %01100111
        dc.b        %01100111
        dc.b        %00000000
; 'l'
        dc.b        %01100000       ;76
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00000000
; 'm'
        dc.b        %01101011       ;77
        dc.b        %01101011
        dc.b        %01101011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'n'
        dc.b        %01101111       ;78
        dc.b        %01100111
        dc.b        %01100111
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'o'
        dc.b        %01100011       ;79
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %00000000
; 'p'
        dc.b        %01111110       ;80
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %01100000
        dc.b        %00000000
; 'q'
        dc.b        %01101011       ;81
        dc.b        %01100111
        dc.b        %01100111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00111110
```

```
        dc.b        %00111110
        dc.b        %00000000
; 'r'
        dc.b        %01111110       ;82
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 's'
        dc.b        %00011110       ;83
        dc.b        %00000011
        dc.b        %00000011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111110
        dc.b        %01111110
        dc.b        %00000000
; 't'
        dc.b        %00001100       ;84
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00001100
        dc.b        %00000000
; 'u'
        dc.b        %01100011       ;85
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00111111
        dc.b        %00111110
        dc.b        %00000000
; 'v'
        dc.b        %01111111       ;86
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %00011100
        dc.b        %00011100
        dc.b        %00001000
        dc.b        %00001000
        dc.b        %00000000
; 'w'
        dc.b        %01111111       ;87
        dc.b        %01101011
        dc.b        %01101011
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %00000000
; 'x'
        dc.b        %00011100       ;88
        dc.b        %00111110
        dc.b        %00111110
        dc.b        %01110111
        dc.b        %01110111
```

```
        dc.b        %01100011
        dc.b        %01100011
        dc.b        %00000000
; 'y'
        dc.b        %00011100       ;89
        dc.b        %00011100
        dc.b        %00011100
        dc.b        %01111000
        dc.b        %01111000
        dc.b        %01110000
        dc.b        %01110000
        dc.b        %00000000
; 'z'
        dc.b        %00111000       ;90
        dc.b        %00111000
        dc.b        %00111000
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %01111111
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;91
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
; ' '
        dc.b        %00000000       ;
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
        dc.b        %00000000
```

```
;
; mancano i caratteri in minuscolo... se avete la pazienza di disegnarli, fate
; pure! Oppure potete fare disegnini da comporre insieme...
;

        SECTION     MIOPLANE,BSS_C      ; Le SECTION BSS devono essere fatte di
                                        ; soli ZERI!!! si usa il DS.b per definire
                                        ; quanti zeri contenga la section.
```

```
BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Questo listato e' uguale a Lezione6c.s, ma il font e' "FATTO A MANO", infatti anziche' caricarlo e' nel listato in forma di dc.b in binario

```
                ;12345678
; "A"
        dc.b        %01111111        ;1
        dc.b        %00000011        ;2
        dc.b        %01100011        ;3
        dc.b        %01111111        ;4
        dc.b        %01100011        ;5
        dc.b        %01100011        ;6
        dc.b        %01100011        ;7
        dc.b        %00000000        ;8
```

Questa per esempio e' la "A". Attenzione a non usare caratteri minuscoli nel testo, perche' non sono nel font, in quanto chi lo ha fatto si deve essere stancato alla "Z" maiuscola. In realta' non c'erano nemmeno molti simboli come "*;<>=" e li ho aggiunti io. Ora apparira' piu' chiara anche come e' fatto il font! E intuirete che per fare un font di 16x16 dovete fare cosi':

```
                ;1234567890123456
; "A"
        dc.w        %0000111111111100        ;1
        dc.w        %0011111111111111        ;2
        dc.w        %0011110000001111        ;3
        dc.w        %0011110000001111        ;4
        dc.w        %0011110000001111        ;5
        dc.w        %0011110000001111        ;6
        dc.w        %0011111111111111        ;7
        dc.w        %0011111111111111        ;8
        dc.w        %0011110000001111        ;9
        dc.w        %0011110000001111        ;10
        dc.w        %0011110000001111        ;11
        dc.w        %0011110000001111        ;12
        dc.w        %0011110000001111        ;13
        dc.w        %0011110000001111        ;14
        dc.w        %0000000000000000        ;15
        dc.w        %0000000000000000        ;16
```

Ma conviene disegnarlo e convertirlo in RAW!

In questo listato vi consiglio di modificare il FONT, aggiungendo disegnini e simboli strani. Potreste farvi il FONT personale!

22.4 Lezione6d

```
; Lezione6d.s                HIRES E LOWRES NELLO STESSO SCHERMO

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
```

```
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l       d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l       d0,a6
        move.l       $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;        Puntiamo i bitplanes in copperlist

        MOVE.L       #PIC+(40*50),d0        ; in d0 mettiamo l'indirizzo della PIC, in
                                 ; questo caso puntiamo piu' avanti di 50
                                 ; linee in modo da far "SALIRE" l'immagine.
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w       d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w       d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w       #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L       #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS2,A1        ; puntatori nella COPPERLIST
        move.w       d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w       d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l       #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w       d0,$dff088                ; Facciamo partire la COP
        move.w       #0,$dff1fc                ; Disattiva l'AGA
        move.w       #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa le linee di testo sullo schermo
                                 ; in HIRES
mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l       gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel (su schermo HIRES)
```

```
PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3         ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #15-1,D3          ; NUMERO RIGHE DA STAMPARE: 15
PRINTRIGA:
        MOVEQ        #80-1,D0          ; NUMERO COLONNE PER RIGA: 80 (hires!)
PRINTCHAR2:
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2         ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2          ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)         ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,80(A3)         ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,80*2(A3)         ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,80*3(A3)         ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,80*4(A3)         ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,80*5(A3)         ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,80*6(A3)         ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,80*7(A3)         ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2         ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #80*7,A3         ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA         ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 80, dunque 2 di queste da 40!
TESTO:           ;                 1111111111222222222233333333334
     ;         12345678901234567890123456789012345678901234567890
        dc.b        '   PRIMA RIGA  IN HIRES 640 PIXEL DI LAR' ; 1a \ PRIMA RIGA
        dc.b        'GHEZZA!  -- -- --   SEMPRE LA PRIMA RIGA' ; 1b /
        dc.b        '               SECONDA RIGA            ' ; 2  \ SECONDA RIGA
        dc.b        'ANCORA SECONDA RIGA                    ' ;    /
        dc.b        '     /\  /                             ' ; 3
        dc.b        '                                       ' ;
        dc.b        '    /  \/                              ' ; 4
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 5
        dc.b        '                                       ' ;
        dc.b        '        SESTA RIGA                     ' ; 6
        dc.b        '                          FINE SESTA RIGA ' ;
        dc.b        '                                       ' ; 7
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 8
        dc.b        '                                       ' ;
        dc.b        'FABIO CIUCCI COMMUNICATION INTERNATIONAL' ; 9
        dc.b        ' MARKETING TRUST TRADEMARK COPYRIGHTED  ' ;
        dc.b        '                                       ' ; 10
```

```
        dc.b        '                                       ' ;
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}     ' ; 11
        dc.b        '   PROVE TECNICHE DI TRASMISSIONE      ' ;
        dc.b        '                                       ' ; 12
        dc.b        '                                       ' ;
        dc.b        '    LA PALINGENETICA OBLITERAZIONE DELL' ; 13
        dc.b       "'IO TRASCENDENTALE CHE SI IMMEDESIMA    " ;
        dc.b        '                                       ' ; 14
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 15
        dc.b        '                                       ' ;
        dc.b        '  Nel mezzo del cammin di nostra vita  ' ; 16
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 17
        dc.b        '                                       ' ;
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa   ' ; 18
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 19
        dc.b        '                                       ' ;
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA    ' ; 20
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 21
        dc.b        '                                       ' ;
        dc.b        '  AHI Quanto a DIR QUAL ERA...         ' ; 22
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 23
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 24
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 25
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 26
        dc.b        '                                       ' ;

        EVEN


        SECTION     GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81       ; DiwStrt       (registri con valori normali)
        dc.w        $90,$2cc1       ; DiwStop
        dc.w        $92,$0038       ; DdfStart
        dc.w        $94,$00d0       ; DdfStop
        dc.w        $102,0              ; BplCon1
        dc.w        $104,0              ; BplCon2
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000      ; bits 13 e 12 accesi!! (3 = %011)

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
```

```
        dc.w        $180,$000        ; color0
        dc.w        $182,$475        ; color1
        dc.w        $184,$fff        ; color2
        dc.w        $186,$ccc        ; color3
        dc.w        $188,$999        ; color4
        dc.w        $18a,$232        ; color5
        dc.w        $18c,$777        ; color6
        dc.w        $18e,$444        ; color7

;        QUA RIDEFINIAMO BPLCON0,COLORI,DDFSTART/STOP e PUNTATORI AI BITPLANES!

        dc.w        $a007,$FFFE

        dc.w        $92,$003c        ; DdfStart HIRES
        dc.w        $94,$00d4        ; DdfStop HIRES
                    ; 5432109876543210
        dc.w        $100,%1001001000000000        ; 1 bitplane HIRES 640x256

BPLPOINTERS2:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
;        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
        incbin        "nice.fnt"        ; Carattere stretto

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,

        SECTION        MIOPLANE,BSS_C        ; Le SECTION BSS devono essere fatte di
                                ; soli ZERI!!! si usa il DS.b per definire
                                ; quanti zeri contenga la section.

BITPLANE:
        ds.b        80*256        ; un bitplane HIres 640x256

        end
```

Si potrebbe cambiare risoluzione grafica anche 50 volte nello stesso schermo, basta ridefinire BPLCON0, colori e puntatori ai bitplanes. Questo puo' essere utile quando servono, ad esempio, 32 colori nel centro dello schermo per far camminare gli ometti di un gioco, ma ne occorrono 16 nel cielo e magari solo 4 per il pannello col punteggio, che si puo' fare in HIRES. Basta considerare che piu' linee in HIRES ci sono e piu' l'esecuzione rallenta, e piu' colori ci sono meno tempo hanno gli altri coprocessori e il 68000 per lavorare. Per questo conviene fare solo delle "strisce" di schermo in HIRES o in HAM, per esempio, lasciando le altre parti con meno colori e in lowres per velocizzare le operazioni.

22.5 Lezione6e

```
; Lezione6e.s                SOVRAPPOSIZIONE DI 2 BITPLANES UGUALI, MA UNO SPOSTATO
```

```
;                       DI UNA LINEA IN BASSO, PER SIMULARE UN FONT OMBREGGIATO

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;        Puntiamo i bitplanes in copperlist

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo del bitplane
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

; NOTATE IL -80!!!!

        MOVE.L        #BITPLANE-80,d0        ; in d0 mettiamo l'indirizzo del bitplane -80
                        ; ossia una linea SOTTO! *******
        LEA        BPLPOINTERS2,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa le linee di testo sullo schermo
                        ; in HIRES
mouse:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel (su schermo HIRES)
```

```
PRINT:
        LEA         TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA         BITPLANE,A3         ; Indirizzo del bitplane destinazione in a3
        MOVEQ         #25-1,D3          ; NUMERO RIGHE DA STAMPARE: 25
PRINTRIGA:
        MOVEQ         #80-1,D0          ; NUMERO COLONNE PER RIGA: 80 (hires!)
PRINTCHAR2:
        MOVEQ         #0,D2                ; Pulisci d2
        MOVE.B         (A0)+,D2         ; Prossimo carattere in d2
        SUB.B         #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W         #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L         D2,A2
        ADD.L         #FONT,A2          ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B         (A2)+,(A3)         ; stampa LA LINEA 1 del carattere
        MOVE.B         (A2)+,80(A3)         ; stampa LA LINEA 2  " "
        MOVE.B         (A2)+,80*2(A3)         ; stampa LA LINEA 3  " "
        MOVE.B         (A2)+,80*3(A3)         ; stampa LA LINEA 4  " "
        MOVE.B         (A2)+,80*4(A3)         ; stampa LA LINEA 5  " "
        MOVE.B         (A2)+,80*5(A3)         ; stampa LA LINEA 6  " "
        MOVE.B         (A2)+,80*6(A3)         ; stampa LA LINEA 7  " "
        MOVE.B         (A2)+,80*7(A3)         ; stampa LA LINEA 8  " "

        ADDQ.w         #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA         D0,PRINTCHAR2         ; STAMPIAMO D0 (80) CARATTERI PER RIGA

        ADD.W         #80*7,A3          ; ANDIAMO A CAPO

        DBRA         D3,PRINTRIGA         ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 80, dunque 2 di queste da 40!
TESTO:           ;                  11111111112222222223333333334
     ;         12345678901234567890123456789012345678901234567890
        dc.b         '   PRIMA RIGA  IN HIRES 640 PIXEL DI LAR' ; 1a \ PRIMA RIGA
        dc.b         'GHEZZA!  -- -- --   SEMPRE LA PRIMA RIGA' ; 1b /
        dc.b         '                SECONDA RIGA            ' ; 2  \ SECONDA RIGA
        dc.b         'ANCORA SECONDA RIGA                     ' ;    /
        dc.b         '     /\  /                              ' ; 3
        dc.b         '                                        ' ;
        dc.b         '    /  \/                               ' ; 4
        dc.b         '                                        ' ;
        dc.b         '                                        ' ; 5
        dc.b         '                                        ' ;
        dc.b         '        SESTA RIGA                      ' ; 6
        dc.b         '                           FINE SESTA RIGA ' ;
        dc.b         '                                        ' ; 7
        dc.b         '                                        ' ;
        dc.b         '                                        ' ; 8
        dc.b         '                                        ' ;
        dc.b         'FABIO CIUCCI COMMUNICATION INTERNATIONAL' ; 9
        dc.b         ' MARKETING TRUST TRADEMARK COPYRIGHTED  ' ;
        dc.b         '                                        ' ; 10
```

```
        dc.b        '                                       ' ;
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}     ' ; 11
        dc.b        '   PROVE TECNICHE DI TRASMISSIONE      ' ;
        dc.b        '                                       ' ; 12
        dc.b        '                                       ' ;
        dc.b        '    LA PALINGENETICA OBLITERAZIONE DELL' ; 13
        dc.b       "'IO TRASCENDENTALE CHE SI IMMEDESIMA    " ;
        dc.b        '                                       ' ; 14
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 15
        dc.b        '                                       ' ;
        dc.b        '  Nel mezzo del cammin di nostra vita  ' ; 16
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 17
        dc.b        '                                       ' ;
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa   ' ; 18
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 19
        dc.b        '                                       ' ;
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA    ' ; 20
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 21
        dc.b        '                                       ' ;
        dc.b        '  AHI Quanto a DIR QUAL ERA...         ' ; 22
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 23
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 24
        dc.b        '                                       ' ;
        dc.b        ' C:\>_                                 ' ; 25
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 26
        dc.b        '                                       ' ;

        EVEN


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$003c        ; DdfStart HIRES
        dc.w        $94,$00d4        ; DdfStop HIRES
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%1010001000000000        ; bit 13 - 2 bitplanes, 4 colori HIRES

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
BPLPOINTERS2:
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
```

```
        dc.w        $180,$103        ; color0 - SFONDO
        dc.w        $182,$fff        ; color1 - plane 1 posizione normale, e'
                                ; la parte che "sporge" in alto.
        dc.w        $184,$345        ; color2 - plane 2 (sfasato in basso)
        dc.w        $186,$abc        ; color3 - entrambi i plane - sovrapposizione

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
;        incbin        "nice.fnt"        ; Carattere stretto


        SECTION        MIOPLANE,BSS_C        ; Le SECTION BSS devono essere fatte di
                                ; soli ZERI!!! si usa il DS.b per definire
                                ; quanti zeri contenga la section.

;        Ecco perche' serve il "ds.b 80":
;        MOVE.L        #BITPLANE-80,d0        ; in d0 mettiamo l'indirizzo del bitplane -80
;                                ; ossia una linea SOTTO! *******

        ds.b        80        ; la linea che "spunta"
BITPLANE:
        ds.b        80*256        ; un bitplane HIres 640x256

        end
```

Ecco qua un "trucchetto" per abbellire la nostra scritta: basta attivare il
secondo bitplane, e sovrapporlo al primo, ma spostato una linea piu' in basso,
in modo da creare questa situazione:

```
        ...###..                        ...111..        ; 1 = color1 (chiaro)
        ..#...#.        ...###..        ..12221.        ; 2 = color2 (scuro)
        ..#...#.        ..#...#.        ..3...3.        ; 3 = color3 (medio)
        ..#####.    +   ..#...#.   =        ..31113.
        ..#...#.        ..#####.        ..32223.
        ..#...#.        ..#...#.        ..3...3.
        ..#...#.        ..#...#.        ..3...3.
        ........        ..#...#.        ..2...2.
                        ........
```

```
        dc.w        $180,$103        ; color0 - SFONDO
        dc.w        $182,$fff        ; color1 - plane 1 posizione normale, e'
                                ; la parte che "sporge" in alto.
        dc.w        $184,$345        ; color2 - plane 2 (sfasato in basso)
        dc.w        $186,$abc        ; color3 - entrambi i plane - sovrapposizione
```

La sovrapposizione di bitplanes uguali, ma sfasati e' usata spesso per simulare
effetti "rilievo" o "spessore"

Per accentuare questo aspetto spesso viene sfasato di un pixel anche in senso
laterale, provate cosi':

```
        dc.w        $102,$10        ; BplCon1 - plane 2 un pixel a destra
```

Su piccoli font forse peggiora la leggibilita', ma su superfici piu' grandi
puo' risultare utile sfasare anche a destra:

```
......
.:::::#
.:::::#
.:::::#
 ######
```

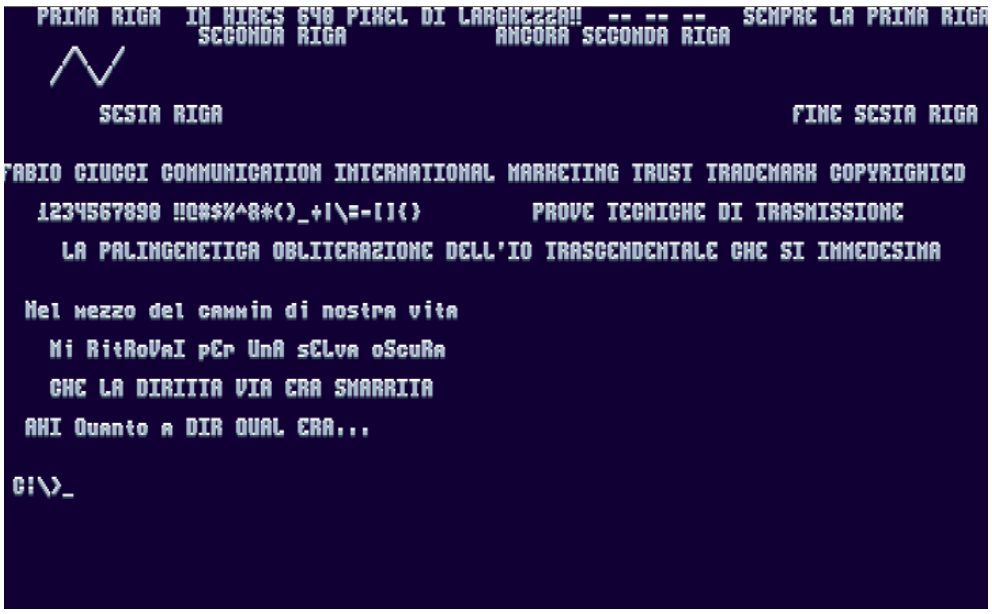


Figura 22.3: Lezione 6e

22.6 Lezione6f

```
; Lezione6f.s                SCRITTURA "SOPRA" UNA FIGURA

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;        Puntiamo i bitplanes in copperlist - prima la PIC

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC, in
                                ; questo caso puntiamo piu' avanti di 50
                                ; linee in modo da far "SALIRE" l'immagine.
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
```

```
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0       ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1               ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP       ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA         BPLPOINTERS2,A1       ; puntatori nella COPPERLIST (plane 4!)
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w         print                ; Stampa le linee di testo sullo schermo
                                  ; in HIRES
mouse:
        btst         #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s         mouse                ; se no, torna a mouse:

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l        gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr         -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                         ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l         0        ; della graphics.library

OldCop:                          ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l         0

;        Routine che stampa caratteri larghi 8x8 pixel (su schermo LOWRES)

PRINT:
        LEA         TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA         BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ         #23-1,D3        ; NUMERO RIGHE DA STAMPARE: 23
PRINTRIGA:
        MOVEQ         #40-1,D0        ; NUMERO COLONNE PER RIGA: 40 (lores)
PRINTCHAR2:
        MOVEQ         #0,D2                ; Pulisci d2
        MOVE.B         (A0)+,D2        ; Prossimo carattere in d2
        SUB.B         #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                  ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                  ; DELLO SPAZIO (che e' $20), in $00, quello
                                  ; DELL'ASTERISCO ($21), in $01...
```

```
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                  ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                  ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #40*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 40
TESTO:           ;                  11111111112222222222333333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b        '                                        ' ; 1
        dc.b        '                SECONDA RIGA            ' ; 2
        dc.b        '     /\  /                              ' ; 3
        dc.b        '    /  \/                               ' ; 4
        dc.b        '                                        ' ; 5
        dc.b        '        SESTA RIGA                      ' ; 6
        dc.b        '                                        ' ; 7
        dc.b        '                                        ' ; 8
        dc.b        'FABIO CIUCCI COMMUNICATION INTERNATIONAL' ; 9
        dc.b        '                                        ' ; 10
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}      ' ; 11
        dc.b        '                                        ' ; 12
        dc.b        ' -=- LA PALINGENETICA OBLITERAZIONE -=- ' ; 13
        dc.b        " ##  DELL'IO TRASCENDENTALE CHE SI  ##  " ; 14
        dc.b        ' ///    IMMEDESIMA E SI INFUTURA   \\\  ' ; 15
        dc.b        '  Nel mezzo del cammin di nostra vita   ' ; 16
        dc.b        '                                        ' ; 17
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa    ' ; 18
        dc.b        '                                        ' ; 19
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA     ' ; 20
        dc.b        '                                        ' ; 21
        dc.b        '  AHI Quanto a DIR QUAL ERA...          ' ; 22
        dc.b        '                                        ' ; 23
        dc.b        '                                        ' ; 24
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 26

        EVEN



        SECTION        GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0100001000000000        ; bit 14 - 4 bitplanes, 16 colori

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
BPLPOINTERS2:
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane

        dc.w        $180,$000        ; color0 ; colori della figura un po' attenuati
        dc.w        $182,$354        ; color1
        dc.w        $184,$678        ; color2
        dc.w        $186,$567        ; color3
        dc.w        $188,$455        ; color4
        dc.w        $18a,$121        ; color5
        dc.w        $18c,$455        ; color6
        dc.w        $18e,$233        ; color7

        dc.w        $190,$d6e        ; color8        ; I colori della scritta:
        dc.w        $192,$d6e        ; color9        ; infatti tutte le varie
        dc.w        $194,$d6e        ; color10        ; sovrapposizioni formerebbero
        dc.w        $196,$d6e        ; color11        ; altri 8 colori, che noi
        dc.w        $198,$d6e        ; color12        ; definiamo tutti uguali
        dc.w        $19a,$d6e        ; color13
        dc.w        $19c,$d6e        ; color14
        dc.w        $19e,$d6e        ; color15

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
;        incbin        "nice.fnt"        ; Carattere stretto

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,

        SECTION        MIOPLANE,BSS_C        ; in CHIP

BITPLANE:
        ds.b        40*256        ; un bitplane lores 320x256

        end
```

```
In questo esempio abbiamo aggiunto il bitplane 4,su cui viene scritto il testo.
Col quarto bitplane si passa da 8 colori a 16, dunque per mantenere la
scrittura dello stesso colore in qualsiasi situazione di sovrapposizione con
gli altri 3 bitplane, tutti i "nuovi" otto colori devono essere uguali al
colore del testo.
Se invece i nuovi 8 colori sono simili ai primi otto, ma piu' chiari, si puo'
ottenere un'effetto di "TRASPARENZA", infatti ogni volta che il plane del testo
si sovrappone agli altri 3, visualizza per ogni pixel dei caratteri un colore
simile a quello della figura ma, appunto, piu' chiaro, e l'illusione della
"TRASPARENZA" avviene automaticamente.
Provate a sostituire la palette degli ultimi colori con questa:

        dc.w        $190,$454        ; color8        ; I colori della scritta:
        dc.w        $192,$7a8        ; color9        ; in questo caso formiamo
        dc.w        $194,$eef        ; color10        ; 8 diversi colori per le
        dc.w        $196,$cde        ; color11        ; 8 possibilita' di
        dc.w        $198,$aab        ; color12        ; sovrapposizione - se notate
        dc.w        $19a,$786        ; color13        ; sono simili ai primi 8,
        dc.w        $19c,$9aa        ; color14        ; ma molto piu' luminosi
        dc.w        $19e,$789        ; color15        ; per creare la "TRASPARENZA"
```

22.7 Lezione6g

```
; Lezione6g.s                SCRITTURA "SOPRA" UNA FIGURA (in trasparenza)
;                          Tasto sinistro del mouse per avanzare, destro per
;                          retrocedere, entrambi per uscire - si puo' scorrere
;                          anche tutta la memoria come in Lezione5l.s

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;        Puntiamo i bitplanes in copperlist - prima la PIC

        MOVE.L        #PIC,d0                ; in d0 mettiamo l'indirizzo della PIC, in
                                ; questo caso puntiamo piu' avanti di 50
                                ; linee in modo da far "SALIRE" l'immagine.
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP        ; Rifai D1 volte POINTBP (D1=num of bitplanes)

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo della PIC,
        LEA        BPLPOINTERS2,A1        ; puntatori nella COPPERLIST (plane 4!)
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
```

```
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w       d0,2(a1)         ; copia la word ALTA dell'indirizzo del plane

        move.l       #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w       d0,$dff088                ; Facciamo partire la COP
        move.w       #0,$dff1fc                ; Disattiva l'AGA
        move.w       #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        print                ; Stampa le linee di testo sullo schermo
                                  ; in HIRES

mouse:
        cmpi.b       #$ff,$dff006        ; Siamo alla linea 255?
        bne.s        mouse               ; Se non ancora, non andare avanti
Aspetta:
        cmpi.b       #$ff,$dff006        ; Siamo alla linea 255?
        beq.s        Aspetta               ; Se si, non andare avanti, aspetta!

        btst        #2,$dff016        ; se il tasto destro e' premuto
        bne.s        NonGiu                ; scorri giu!, oppure vai a NonGiu

        bsr.w        VaiGiu                ; tasto destro premuto, scorri giu!

Nongiu:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        beq.s        Scorrisu        ; se si, scorri in su
        bra.s        mouse                ; no? allora ripeti il ciclo il prossimo FRAME

Scorrisu:
        bsr.w        VaiSu                ; fa scorrere la figura in alto

        btst        #2,$dff016        ; se anche il tasto destro e' premuto allora
        bne.s        mouse                ; sono premuti entrambi, esci, oppure "MOUSE"


        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l       gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr        -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                        ; USCITA DAL PROGRAMMA

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l        0        ; della graphics.library

OldCop:                        ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l        0

;        Routine che stampa caratteri larghi 8x8 pixel (su schermo LOWRES)

PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #23-1,D3        ; NUMERO RIGHE DA STAMPARE: 23
PRINTRIGA:
```

```
        MOVEQ        #40-1,D0        ; NUMERO COLONNE PER RIGA: 40 (lores)
PRINTCHAR2:
        MOVEQ        #0,D2               ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        SUB.B        #$20,D2               ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2               ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3               ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #40*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 40
TESTO:             ;                 1111111111222222222233333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b        '                                        ' ; 1
        dc.b        '                SECONDA RIGA            ' ; 2
        dc.b        '     /\  /                              ' ; 3
        dc.b        '    /  \/                               ' ; 4
        dc.b        '                                        ' ; 5
        dc.b        '        SESTA RIGA                      ' ; 6
        dc.b        '                                        ' ; 7
        dc.b        '                                        ' ; 8
        dc.b        'FABIO CIUCCI COMMUNICATION INTERNATIONAL' ; 9
        dc.b        '                                        ' ; 10
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}      ' ; 11
        dc.b        '                                        ' ; 12
        dc.b        ' -=- LA PALINGENETICA OBLITERAZIONE -=- ' ; 13
        dc.b        " ##  DELL'IO TRASCENDENTALE CHE SI  ##  " ; 14
        dc.b        ' ///    IMMEDESIMA E SI INFUTURA   \\\  ' ; 15
        dc.b        '  Nel mezzo del cammin di nostra vita   ' ; 16
        dc.b        '                                        ' ; 17
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa    ' ; 18
        dc.b        '                                        ' ; 19
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA     ' ; 20
        dc.b        '                                        ' ; 21
        dc.b        '  AHI Quanto a DIR QUAL ERA...          ' ; 22
        dc.b        '                                        ' ; 23
        dc.b        '                                        ' ; 24
        dc.b        '                                        ' ; 25
```

```
        dc.b        '                                        ' ; 26

        EVEN

;        Questa routine sposta la figura in alto e in basso, agendo sui
;        puntatori ai bitplanes in copperlist (tramite la label BPLPOINTERS)
;        Da Lezione5l.s

VAIGIU:
        LEA         BPLPOINTERS2,A1         ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0          ; copperlist l'indirizzo dove sta puntando
        swap         d0                 ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0          ; in d0 - il contrario della routine che
        sub.l         #40,d0                ; sottraiamo 40, ossia 1 linea, facendo
                                ; scorrere in BASSO la figura
        bra.s         Finito


VAISU:
        LEA         BPLPOINTERS2,A1         ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0          ; copperlist l'indirizzo dove sta puntando
        swap         d0                 ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0          ; in d0 - il contrario della routine che
        add.l         #40,d0                ; Aggiungiamo 40, ossia 1 linea, facendo
                                ; scorrere in ALTO la figura
        bra.w         finito


Finito:                                  ; PUNTIAMO I PUNTATORI BITPLANES
        move.w        d0,6(a1)          ; copia la word BASSA dell'indirizzo del plane
        swap         d0                 ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)          ; copia la word ALTA dell'indirizzo del plane
        rts



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w         $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w         $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w         $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w         $13e,$0000

        dc.w         $8e,$2c81          ; DiwStrt        (registri con valori normali)
        dc.w         $90,$2cc1          ; DiwStop
        dc.w         $92,$0038          ; DdfStart
        dc.w         $94,$00d0          ; DdfStop
        dc.w         $102,0                  ; BplCon1
        dc.w         $104,0                  ; BplCon2
        dc.w         $108,0                  ; Bpl1Mod
        dc.w         $10a,0                  ; Bpl2Mod

                     ; 5432109876543210
        dc.w         $100,%0100001000000000        ; bit 14 - 4 bitplanes, 16 colori

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
BPLPOINTERS2:
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane
```

```
        dc.w        $180,$000        ; color0 ; colori della figura un po' attenuati
        dc.w        $182,$354        ; color1
        dc.w        $184,$678        ; color2
        dc.w        $186,$567        ; color3
        dc.w        $188,$455        ; color4
        dc.w        $18a,$121        ; color5
        dc.w        $18c,$455        ; color6
        dc.w        $18e,$233        ; color7

        dc.w        $190,$454        ; color8        ; I colori della scritta:
        dc.w        $192,$7a8        ; color9        ; in questo caso formiamo
        dc.w        $194,$eef        ; color10        ; 8 diversi colori per le
        dc.w        $196,$cde        ; color11        ; 8 possibilita' di
        dc.w        $198,$aab        ; color12        ; sovrapposizione - se notate
        dc.w        $19a,$786        ; color13        ; sono simili ai primi 8,
        dc.w        $19c,$9aa        ; color14        ; ma molto piu' luminosi
        dc.w        $19e,$789        ; color15        ; per creare la "TRASPARENZA"

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
;        incbin        "nice.fnt"        ; Carattere stretto

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,

        SECTION        MIOPLANE,BSS_C        ; in CHIP

BITPLANE:
        ds.b        40*256        ; un bitplane lores 320x256

        end

Potete scorrere anche tutta la memoria sopra la figura! Se indietreggiate col
tasto destro del mouse troverete i 3 bitplanes della figura, poi il font
caratteri (non ben visibile per l'incongruenza di modulo) eccetera.
```

22.8 Lezione6h

```
; Lezione6h.s        STAMPIAMO VARIE RIGHE DI TESTO * A 3 COLORI * ATTIVANDO IL
;                SECONDO BITPLANE, SU CUI SCRIVIAMO IL TESTO2.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase in a6
        jsr        -$78(a6)        ; Disable - ferma il multitasking
        lea        GfxName(PC),a1        ; Indirizzo del nome della lib da aprire in a1
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase        ; salvo l'indirizzo base GFX in GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo l'indirizzo della copperlist vecchia

;         PUNTIAMO I NOSTRI BITPLANE
```

```
        MOVE.L      #BITPLANE,d0        ; in d0 mettiamo l'indirizzo del bitplane1
        LEA       BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w      d0,6(a1)          ; copia la word BASSA dell'indirizzo del plane
        swap       d0                 ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w      d0,2(a1)          ; copia la word ALTA dell'indirizzo del plane

        MOVE.L      #BITPLANE2,d0       ; in d0 mettiamo l'indirizzo del bitplane 2
        LEA       BPLPOINTERS2,A1       ; puntatori nella COPPERLIST
        move.w      d0,6(a1)          ; copia la word BASSA dell'indirizzo del plane
        swap       d0                 ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w      d0,2(a1)          ; copia la word ALTA dell'indirizzo del plane

        move.l      #COPPERLIST,$dff080       ; Puntiamo la nostra COP
        move.w      d0,$dff088              ; Facciamo partire la COP
        move.w      #0,$dff1fc              ; Disattiva l'AGA
        move.w      #$c00,$dff106              ; Disattiva l'AGA

        LEA       TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA       BITPLANE,A3         ; Indirizzo del bitplane destinazione in a3
        bsr.w       print               ; Stampa le linee di testo sullo schermo

        LEA       TESTO2(PC),A0       ; Indirizzo del testo da stampare in a0
        LEA       BITPLANE2,A3        ; Indirizzo del bitplane destinazione in a3
        bsr.w       print               ; Stampa le linee di testo sullo schermo

mouse:
        btst       #6,$bfe001         ; tasto sinistro del mouse premuto?
        bne.s       mouse              ; se no, torna a mouse:

        move.l      OldCop(PC),$dff080       ; Puntiamo la cop di sistema
        move.w      d0,$dff088              ; facciamo partire la vecchia cop

        move.l      4.w,a6
        jsr       -$7e(a6)        ; Enable - riabilita il Multitasking
        move.l      gfxbase(PC),a1        ; Base della libreria da chiudere
        jsr       -$19e(a6)        ; Closelibrary - chiudo la graphics lib
        rts                       ; USCITA DAL PROGRAMMA

;       Dati

GfxName:
        dc.b       "graphics.library",0,0

GfxBase:                ; Qua ci va l'indirizzo di base per gli Offset
        dc.l       0        ; della graphics.library

OldCop:                       ; Qua ci va l'indirizzo della vecchia COP di sistema
        dc.l       0

;       Routine che stampa caratteri larghi 8x8 pixel

PRINT:
        MOVEQ       #23-1,D3        ; NUMERO RIGHE DA STAMPARE: 23
PRINTRIGA:
        MOVEQ       #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
        MOVEQ       #0,D2               ; Pulisci d2
        MOVE.B       (A0)+,D2       ; Prossimo carattere in d2
        SUB.B       #$20,D2               ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                  ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                  ; DELLO SPAZIO (che e' $20), in $00, quello
                                  ; DELL'ASTERISCO ($21), in $01...
```

```
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                  ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                  ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #40*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 40
TESTO:           ;                  1111111111222222222233333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b        '   PRIMA RIGA (solo in testo1)          ' ; 1
        dc.b        '                                        ' ; 2
        dc.b        '     /\  /                              ' ; 3
        dc.b        '    /  \/                               ' ; 4
        dc.b        '                                        ' ; 5
        dc.b        '        SESTA RIGA (entrambi i bitplane)' ; 6
        dc.b        '                                        ' ; 7
        dc.b        '                                        ' ; 8
        dc.b        'FABIO CIUCCI               INTERNATIONAL' ; 9
        dc.b        '                                        ' ; 10
        dc.b        '   1  4 6 89  !@ $ ^& () +| =- ]{       ' ; 11
        dc.b        '                                        ' ; 12
        dc.b        '     LA  A I G N T C  OBLITERAZIONE     ' ; 15
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 16
        dc.b        '  Nel mezzo del cammin di nostra vita   ' ; 17
        dc.b        '                                        ' ; 18
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa    ' ; 19
        dc.b        '                                        ' ; 20
        dc.b        '    CHE LA DIRITTA VIA ERA              ' ; 21
        dc.b        '                                        ' ; 22
        dc.b        '  AHI Quanto a DIR QUAL ERA...          ' ; 23
        dc.b        '                                        ' ; 24
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 26
        dc.b        '                                        ' ; 27

        EVEN

              ; numero caratteri per linea: 40
TESTO2:          ;                  1111111111222222222233333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b        '                                        ' ; 1
```

```
        dc.b        '  SECONDA RIGA (solo in testo2)         ' ; 2
        dc.b        '     /\  /                              ' ; 3
        dc.b        '    /  \/                               ' ; 4
        dc.b        '                                        ' ; 5
        dc.b        '        SESTA RIGA (entrambi i bitplane)' ; 6
        dc.b        '                                        ' ; 7
        dc.b        '                                        ' ; 8
        dc.b        'FABIO        COMMUNICATION INTERNATIONAL' ; 9
        dc.b        '                                        ' ; 10
        dc.b        '   1234567 90  @#$%^&*( _+|\=-[]{}      ' ; 11
        dc.b        '                                        ' ; 12
        dc.b        '    LA PALINGENETICA  B I E A I N       ' ; 15
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 16
        dc.b        '  Nel       del cammin di        vita   ' ; 17
        dc.b        '                                        ' ; 18
        dc.b        '    Mi          pEr UnA       oScuRa    ' ; 19
        dc.b        '                                        ' ; 20
        dc.b        '    CHE LA         VIA ERA SMARRITA     ' ; 21
        dc.b        '                                        ' ; 22
        dc.b        '  AHI Quanto a     QUAL ERA...          ' ; 23
        dc.b        '                                        ' ; 24
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 26
        dc.b        '                                        ' ; 27

        EVEN



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,$0000,$122,$0000,$124,$0000,$126,$0000,$128,$0000 ; SPRITE
        dc.w        $12a,$0000,$12c,$0000,$12e,$0000,$130,$0000,$132,$0000
        dc.w        $134,$0000,$136,$0000,$138,$0000,$13a,$0000,$13c,$0000
        dc.w        $13e,$0000

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0010001000000000        ; 2 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
BPLPOINTERS2:
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE primo bitplane
        dc.w        $0184,$f62        ; color2 - SCRITTE secondo bitplane
        dc.w        $0186,$1e4        ; color3 - SCRITTE primo+secondo bitplane

        dc.w        $FFFF,$FFFE       ; Fine della copperlist

;        Il FONT caratteri 8x8
```

```
FONT:
;        incbin        "metal.fnt"        ; Carattere largo
;        incbin        "normal.fnt"        ; Simile ai caratteri kickstart 1.3
        incbin        "nice.fnt"        ; Carattere stretto

        SECTION        MIOPLANE,BSS_C        ; In CHIP

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256
BITPLANE2:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Per fare il testo a 3 colori (4 con lo sfondo) e' bastato attivare un altro bitplane e stamparci il testo2, modificato in modo che manchino delle parole per creare colori diversi nella sovrapposizione. Anche facendo "mancare" delle parole al testo del primo bitplane si cambia il colore, infatti rimane solo il secondo bitplane. Per stampare entrambi i testi si usa la stessa routine print, ma c'e' una piccola modifica: le prime due istruzioni, che prelevano gli indirizzi del testo da stampare e del bitplane destinazione sono state tolte, in modo da rendere la routine utilizzabile per stampare qualsiasi testo preventivamente caricato in a0 nel bitplane preventivamente caricato in a3:

```
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        bsr.w        print                ; Stampa le linee di testo sullo schermo

        LEA        TESTO2(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE2,A3        ; Indirizzo del bitplane destinazione in a3
        bsr.w        print                ; Stampa le linee di testo sullo schermo
```

In questo modo la routine print puo' essere utilizzata per stampare qualsiasi testo su qualsiasi bitplane, e non sempre TESTO: in BITPLANE:!
Al primo "bsr.w print" stampa come nei listati precedenti, al secondo bsr.w invece stampa il TESTO2: nel BITPLANE2:.
Dalla sovrapposizione dei 2 bitplane, a seconda che il carattere sia solo nel primo bitplane, o sia solo nel secondo, o sia in entrambi, viene visualizzato uno dei 3 colori (il quarto e' lo sfondo)

```
        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE primo bitplane (BLU)
        dc.w        $0184,$f62        ; color2 - SCRITTE secondo bitplane (ARANCIO)
        dc.w        $0186,$1e4        ; color3 - SCRITTE primo+secondo bitpl. (VERDE)
```

Per vedere meglio la situazione dei 2 biplane provate a spostare in alto il secondo bitplane di 5 pixel:

```
        MOVE.L        #BITPLANE2+(40*5),d0
```

22.9 Lezione6i

```
; Lezione6i.s        TESTO A 3 COLORI CON UN COLORE LAMPEGGIANTE OTTENUTO USANDO
;                UNA TABELLA DI COLORI RGB PREFISSATI.


        SECTION        CiriCop,CODE        ; NOTA: ho tolto alcuni commenti iniziali per
```

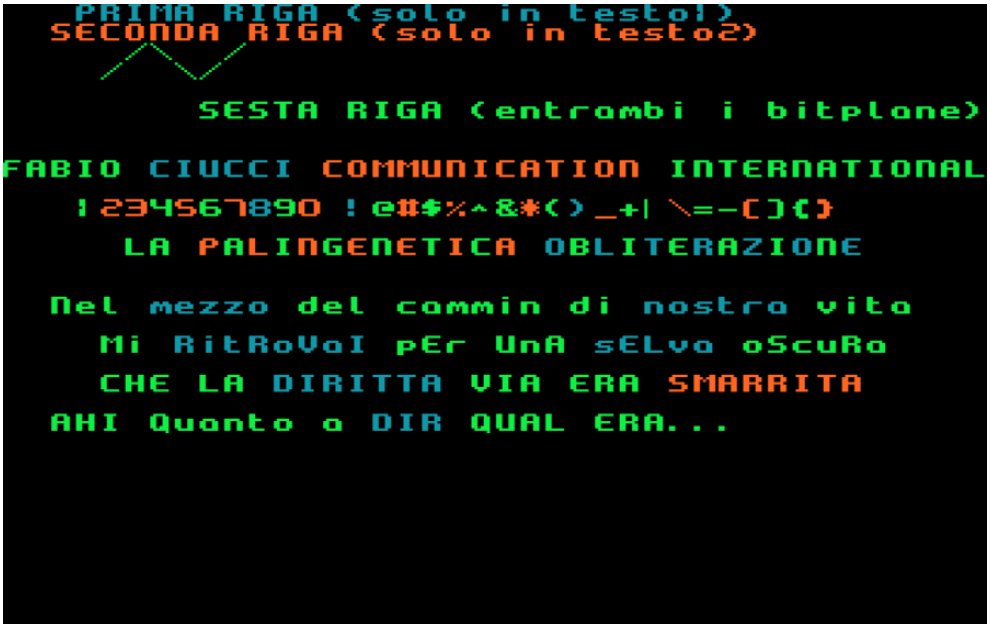


Figura 22.4: Lezione 6h

```
                                ; risparmiare spazio!

Inizio:
        move.l      4.w,a6                ; Execbase
        jsr         -$78(a6)          ; Disable
        lea         GfxName(PC),a1          ; Nome lib
        jsr         -$198(a6)          ; OpenLibrary
        move.l      d0,GfxBase
        move.l      d0,a6
        move.l      $26(a6),OldCop          ; salviamo la vecchia COP

        MOVE.L      #BITPLANE,d0          ; dove puntare
        LEA         BPLPOINTERS,A1        ; puntatori COP
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        MOVE.L      #BITPLANE2,d0         ; dove puntare
        LEA         BPLPOINTERS2,A1       ; puntatori COP
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        move.l      #COPPERLIST,$dff080         ; nostra COP
        move.w      d0,$dff088                ; START COP
        move.w      #0,$dff1fc                ; NO AGA!
        move.w      #$c00,$dff106                ; NO AGA!

        LEA         TESTO(PC),A0          ; testo da stampare
        LEA         BITPLANE,A3         ; destinazione
        bsr.w       print                 ; Stampa
```

```
        LEA        TESTO2(PC),A0        ; testo da stampare
        LEA        BITPLANE2,A3        ; destinazione
        bsr.w        print                ; Stampa

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        btst        #2,$dff016        ; tasto destro?
        beq.s        aspetta

        bsr.w        Lampeggio        ; Fa lampeggiare il Color2 in copperlist

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

;        Routine di lampeggiamento che sfrutta una TABELLA di sfumature gia'
;        pronte. La TABELLA non e' altro che una serie di words contenenti
;        i vari valori RGB che il COLOR1 dovra' assumere nei vari fotogrammi.
;        Questa routine copia la word successiva nella tabella ogni volta che
;        viene eseguita, e una volta raggiunta l'ultima word della TABELLA,
;        ossia 1 word (2 bytes) prima della label FINECOLORTAB, riparte la
;        lettura della tabella dall'inizio, ad esempio:
;
;        dc.w        1,3,5,7,9,8,6,4,2,1        ; la nostra "mini" tabella
;
;        Durante i vari fotogrammi le word saranno copiate, all'infinito, con
;        questo ordine:
;
;        1,3,5,7,9,8,6,4,2,1,1,3,5,7,9,8,6,4,2,1,1,3,5,7,9,8,6,4,2,1....
;
;        L'indirizzo dell'ultima word letta viene tenuto nella long COLTABPOINT

Lampeggio:
        ADDQ.L        #2,COLTABPOINT        ; Fai puntare alla word successiva
        MOVE.L        COLTABPOINT(PC),A0 ; indirizzo contenuto in long COLTABPOINT
                                ; copiato in a0
        CMP.L        #FINECOLORTAB-2,A0 ; Siamo arrivati all'ultima word della TAB?
```

```
        BNE.S       NOBSTART2               ; non ancora? allora continua
        MOVE.L       #COLORTAB-2,COLTABPOINT        ; Riparti a puntare dalla prima word
NOBSTART2:
        MOVE.W       (A0),COLORE1        ; copia la word dalla tabella al colore COP
        rts


COLTABPOINT:                        ; Questa longword "PUNTA" a COLORTAB, ossia
        dc.l        COLORTAB-2        ; contiene l'indirizzo di COLORTAB. Terra'
                                ; l'indirizzo del'ultima word "letta" dentro
                                ; la tabella. (qua inizia da COLORTAB-2 in
                                ; quanto Lampeggio inizia con un ADDQ.L #2,C..
                                ; serve per "bilanciare" la prima istruzione.

;        La tabella con i valori "precalcolati" del lampeggiamento di color0

COLORTAB:
        dc.w        $000,$000,$001,$011,$011,$011,$012,$012        ; inizio SCURO
        dc.w        $022,$022,$022,$023,$023
        dc.w        $033,$033,$034
        dc.w        $044,$044
        dc.w        $045,$055,$055
        dc.w        $056,$056,$066,$066,$066
        dc.w        $167,$167,$177,$177,$177,$177,$177
        dc.w        $278,$278,$278,$288,$288,$288,$288,$288
        dc.w        $389,$389,$399,$399,$399,$399
        dc.w        $39a,$39a,$3aa,$3aa,$3aa
        dc.w        $3ab,$3bb,$3bb,$3bb
        dc.w        $4bc,$4cc,$4cc,$4cc
        dc.w        $4cd,$4cd,$4dd,$4dd,$4dd
        dc.w        $5de,$5de,$5ee,$5ee,$5ee,$5ee
        dc.w        $6ef,$6ff,$6ff,$7ff,$7ff,$8ff,$8ff,$9ff        ; ,massimo CHIARO
        dc.w        $5ee,$5ee,$5ee,$5de,$5de,$5de
        dc.w        $4dd,$4dd,$4dd,$4cd,$4cd
        dc.w        $4cc,$4cc,$4cc,$4bc
        dc.w        $3cb,$3bb,$3bb
        dc.w        $3ba,$3aa,$3aa
        dc.w        $3a9,$399,$399
        dc.w        $298,$288
        dc.w        $187,$177
        dc.w        $076,$066
        dc.w        $065,$055
        dc.w        $054,$044
        dc.w        $034
        dc.w        $022
        dc.w        $011
        dc.w        $000                        ; di nuovo SCURO
FINECOLORTAB:

;        Routine che stampa caratteri larghi 8x8 pixel

PRINT:
        MOVEQ        #23-1,D3        ; NUMERO RIGHE DA STAMPARE: 23
PRINTRIGA:
        MOVEQ        #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...
```

```
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W        #40*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE

        RTS


              ; numero caratteri per linea: 40
TESTO:           ;                  1111111111222222222233333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b        '   PRIMA RIGA (solo in testo1)          ' ; 1
        dc.b        '                                        ' ; 2
        dc.b        '     /\  /                              ' ; 3
        dc.b        '    /  \/                               ' ; 4
        dc.b        '                                        ' ; 5
        dc.b        '        SESTA RIGA (entrambi i bitplane)' ; 6
        dc.b        '                                        ' ; 7
        dc.b        '                                        ' ; 8
        dc.b        'FABIO CIUCCI               INTERNATIONAL' ; 9
        dc.b        '                                        ' ; 10
        dc.b        '   1  4 6 89  !@ $ ^& () +| =- ]{       ' ; 11
        dc.b        '                                        ' ; 12
        dc.b        '     LA  A I G N T C  OBLITERAZIONE     ' ; 15
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 16
        dc.b        '  Nel mezzo del cammin di nostra vita   ' ; 17
        dc.b        '                                        ' ; 18
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa    ' ; 19
        dc.b        '                                        ' ; 20
        dc.b        '    CHE LA DIRITTA VIA ERA              ' ; 21
        dc.b        '                                        ' ; 22
        dc.b        '  AHI Quanto a DIR QUAL ERA...          ' ; 23
        dc.b        '                                        ' ; 24
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 26
        dc.b        '                                        ' ; 27

        EVEN

              ; numero caratteri per linea: 40
TESTO2:          ;                  1111111111222222222233333333334
            ;         12345678901234567890123456789012345678901234567890
        dc.b        '                                        ' ; 1
        dc.b        '  SECONDA RIGA (solo in testo2)         ' ; 2
        dc.b        '     /\  /                              ' ; 3
        dc.b        '    /  \/                               ' ; 4
        dc.b        '                                        ' ; 5
        dc.b        '        SESTA RIGA (entrambi i bitplane)' ; 6
        dc.b        '                                        ' ; 7
```

```
        dc.b        '                                        ' ; 8
        dc.b        'FABIO        COMMUNICATION INTERNATIONAL' ; 9
        dc.b        '                                        ' ; 10
        dc.b        '   1234567 90  @#$%^&*( _+|\=-[]{}      ' ; 11
        dc.b        '                                        ' ; 12
        dc.b        '     LA PALINGENETICA  B I E A I N      ' ; 15
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 16
        dc.b        '  Nel       del cammin di        vita   ' ; 17
        dc.b        '                                        ' ; 18
        dc.b        '    Mi          pEr UnA       oScuRa    ' ; 19
        dc.b        '                                        ' ; 20
        dc.b        '    CHE LA         VIA ERA SMARRITA     ' ; 21
        dc.b        '                                        ' ; 22
        dc.b        '  AHI Quanto a     QUAL ERA...          ' ; 23
        dc.b        '                                        ' ; 24
        dc.b        '                                        ' ; 25
        dc.b        '                                        ' ; 26
        dc.b        '                                        ' ; 27

        EVEN


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0010001000000000        ; 2 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane
BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ;secondo bitplane

        dc.w        $180,$000        ; color0 - SFONDO
        dc.w        $182
COLORE1:
        dc.w        $000               ; color1 - SCRITTE primo bitplane (blu)
        dc.w        $184,$f62        ; color2 - SCRITTE secondo bitplane (arancio)
        dc.w        $186,$1e4        ; color3 - SCRITTE primo+secondo bitpl. (verde)

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"
;        incbin        "normal.fnt"
;        incbin        "nice.fnt"
```

```
        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; lowres 320x256
BITPLANE2:
        ds.b        40*256        ; lowres 320x256

        end
```

Con l'uso di valori predeterminati, o "precalcolati" si possono ottenere
effetti di movimento o di sfumatura dei colori molto migliori che tramite
l'uso di soli ADD e SUB.
L'unica "novita'" e' la tecnica di programmazione della routine "Lampeggio" che
legge i valori da mettere nel "COLORE1" da una tabella, in cui viene usato
un PUNTATORE all'ultima word letta, ossia una LONGWORD che contiene l'indirizzo
di quella word nella tabella. Da notare che un:

```
COLTABPOINT:
        dc.l        COLORTAB
```

E' come un

```
COLTABPOINT:
        DC.L        0
```

Dopo un MOVE.L #COLORTAB,COLTABPOINT, ossia viene assemblata una longword che
contiene l'indirizzo della label in questione. In questa routine c'e' un

```
        dc.l        COLORTAB-2
```

Ma e' soltanto per far leggere la prima word la prima volta, dato che la
routine comincia con:

```
Lampeggio:
        ADDQ.L        #2,COLTABPOINT        ; Fai puntare alla word successiva
```

Bisogna che COLTABPOINT contenga l'inizio della prima word-2, almeno dopo il
primo ADDQ.L #2 al primo jsr viene copiata la prima word e non la seconda.
Successivamente la longword COLTABPOINT sara' aumentata di 2 ogni volta, ossia
l'indirizzo che contiene sara' quello delle varie word, fino a che non
raggiungera' l'ultima word, che comincia 2 bytes prima della fine della
tabella:

```
; siamo a questo indirizzo quando leggiamo l'ultima word...
        dc.w        $0000                        ; di nuovo SCURO
FINECOLORTAB:
```

A questo punto con un:

```
        MOVE.L        #COLORTAB,COLTABPOINT        ; Riparti a puntare dalla prima word
```

la label COLTABPOINT ritorna a contenere l'indirizzo della prima word.

Potete usare questa routine cambiando la tabella per moltissimi scopi, per
esempio per far saltellare o ondeggiare uno sprite.

Provate a sostituire la tabella con questa:

```
COLORTAB:
        dc.w        $26F,$27E,$28D,$29C,$2AB,$2BA,$2C9,$2D8,$2E7,$2F6
```

```
        dc.w        $4E7,$6D8,$8C9,$ABA,$CAA,$D9A,$E8A,$F7A,$F6B,$F5C
        dc.w        $D6D,$B6E,$96F,$76F,$56F,$36F
FINECOLORTAB:
```

22.10 Lezione6l

```
; Lezione6l.s        COLORE LAMPEGGIANTE TRAMITE L'USO DI UNA TABELLA con routine
;                 che una volta finita la tabella la rilegge all'indietro


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        btst        #2,$dff016        ; tasto destro?
        beq.s        aspetta

        bsr.w        Lampeggio        ; Fa lampeggiare il Color0 in copperlist

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
```

```
        dc.l        0

;        Routine di lampeggiamento che sfrutta una TABELLA di sfumature gia'
;        pronte. La TABELLA non e' altro che una serie di words contenenti
;        i vari valori RGB che il COLOR1 dovra' assumere nei vari fotogrammi.
;        Da notare che la tabella viene "letta" in questo modo: si parte col
;        copiare la prima word, poi ogni volta che viene rieseguita nei
;        fotogrammi seguenti la routine copia la seconda word, la terza, la
;        quarta e cosi' via, fino a che non giunge all'ultimo valore della
;        tabella, alla label FINECOLORTAB, allora inverte la direzione col
;        BCHG.B        #1,DIREZFLAG, e procede leggendo "all'indietro" ogni volta
;        fino a che non torna alla prima word, allora cambia ancora DIREZFLAG
;        e riprende leggendo "in avanti".
;        NOTA: questa routine e' utile quando i valori in "aumento" della
;        tabella, una volta raggiunto il "massimo", calano in maniera uguale
;        a come sono aumentati: in questo caso, avremmo dovuto scrivere
;        una tabella di questo tipo:
;
;        dc.w        0,1,2,3,4,5,6,7,8,9,10 ; progressione fino al massimo
;        dc.w        10,9,8,7,6,5,4,3,2,1,0 ; calo fono al minimo
;
;        Ma con questa routine si puo' scrivere la sola meta' della TABELLA,
;        ossia fino al 10, sara' la routine a "tornare indietro" una volta
;        raggiunto il massimo: 9,8,7,6,5,4..., risparmiando spazio nel
;        listato, e tempo se i valori sono scritti "a mano".
;        Se la tabella invece non era "speculare", ossia di questo tipo:
;
;        dc.b        0,2,3,5,6,7,8,9,10
;        dc.b        9,8,7,6,4,3,2,1,0
;
;        Sarebbe stata usata una routine che legge tutta la tabella, dal primo
;        valore all'ultimo, ma che anziche' rileggere all'indietro una volta
;        raggiunto il termine, ricominci da capo.

Lampeggio:
        BTST        #1,DIREZFLAG        ; dobbiamo leggere in avanti all'indietro le
        BEQ.S        GIUT2                ; word della tabella??
SUT2:
        SUBQ.L        #2,COLTABPOINT        ; Fai puntare alla word precedente
        MOVE.L        COLTABPOINT(PC),A0 ; indirizzo contenuto in long COLTABPOINT
                                ; copiato in a0
        CMP.L        #COLORTAB,A0        ; Siamo arrivati al primo valore della TABELLA?
        BNE.S        NOBSTART2
        BCHG.B        #1,DIREZFLAG        ; cambia direzione, vai in avanti!
NOBSTART2:
        MOVE.W        (A0),COLORE0        ; copia la word dalla tabella al colore COP
        rts

GIUT2:
        ADDQ.L        #2,COLTABPOINT           ; Fai puntare alla prossima word
        MOVE.L        COLTABPOINT(PC),A0 ; Indirizzo in COLTABPOINT copiato in a0
        CMP.L        #FINECOLORTAB-2,A0 ; Siamo all'ultima word della TAB?
        BNE.S        NONCAMBDIREZ           ; Se non ancora, non cambiare nulla
        BCHG.B        #1,DIREZFLAG           ; cambia direzione, vai all'indietro!
NONCAMBDIREZ:
        MOVE.W        (A0),COLORE0        ; coopia la word dalla tabella al colore COP
        rts

DIREZFLAG:                        ; Label FLAG usata per indicare la direzione
        DC.W        0                ; di lettura.
```

```
COLTABPOINT:                        ; Questa longword "PUNTA" a COLORTAB, ossia
        dc.l        COLORTAB-2        ; contiene l'indirizzo di COLORTAB. Terra'
                                ; l'indirizzo dell'ultima word "letta" dentro
                                ; la tabella.

;        La tabella con i valori "precalcolati" del lampeggiamento di color0

COLORTAB:
        dc.w        $000,$000,$001,$011,$011,$011,$012,$012        ; inizio SCURO
        dc.w        $022,$022,$022,$023,$023
        dc.w        $033,$033,$034
        dc.w        $044,$044
        dc.w        $045,$055,$055
        dc.w        $056,$056,$066,$066,$066
        dc.w        $167,$167,$177,$177,$177,$177,$177
        dc.w        $278,$278,$278,$288,$288,$288,$288,$288
        dc.w        $389,$389,$399,$399,$399,$399
        dc.w        $39a,$39a,$3aa,$3aa,$3aa
        dc.w        $3ab,$3bb,$3bb,$3bb
        dc.w        $4bc,$4cc,$4cc,$4cc
        dc.w        $4cd,$4cd,$4dd,$4dd,$4dd
        dc.w        $5de,$5de,$5ee,$5ee,$5ee,$5ee
        dc.w        $6ef,$6ff,$6ff,$7ff,$7ff,$8ff,$8ff,$9ff        ; ,massimo CHIARO
FINECOLORTAB:


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0000001000000000        ; 0 bitplane LOWRES

        dc.w        $180,$000        ; color0

        dc.w        $a007,$fffe        ; Wait linea $a0
        dc.w        $180
COLORE0:
        dc.w        $000        ; color0

        dc.w        $c007,$fffe        ;Wait linea $c0
        dc.w        $180,$000        ; color0

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

        end
```

Questa e' una delle tante varianti della routine che legge i valori da una tabella. Questa routine puo' essere usata solo nei casi di tabelle "a specchio" ossia con valori crescenti uguali a quelli decrescenti con il "massimo"

```
raggiunto proprio nel mezzo.
L'effetto infatti e' piu' simmetrico di quello in Lezione6i.s

Provate a cambiare TABELLA e cambiera' tutto: (Amiga+b+c+i)

COLORTAB:
        dc.w $000,$100,$200,$300,$400,$500,$600,$700
        dc.w $800,$900,$a00,$b00,$c00,$d00,$e00
        dc.w $f00,$f10,$f20,$f30,$f40,$f50,$f60,$f70
        dc.w $f80,$f90,$fa0,$fb0,$fc0,$fd0,$fe0
        dc.w $ff0,$ef0,$df0,$cf0,$bf0,$af0,$9f0,$8f0
        dc.w $7f0,$6f0,$5f0,$4f0,$3f0,$2f0,$1f0
        dc.w $0f0,$0f1,$0f2,$0f3,$0f4,$0f5,$0f6,$0f7
        dc.w $0f8,$0f9,$0fa,$0fb,$0fc,$0fd,$0fe
        dc.w $0ff,$0ef,$0df,$0cf,$0bf,$0af,$09f,$08f
        dc.w $07f,$06f,$05f,$04f,$03f,$02f,$01f
        dc.w $00f,$10f,$20f,$30f,$40f,$50f,$60f,$70f
        dc.w $80f,$90f,$a0f,$b0f,$c0f,$d0f,$e0f
        dc.w $f0f,$e0e,$d0d,$c0c,$b0b,$a0a,$909,$808
        dc.w $707,$606,$505,$404,$303,$202,$101,$000
FINECOLORTAB:

Provate anche questa TABELLA:

COLORTAB:
        dc.w 0,0,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
        dc.w $10f,$20f,$30f,$40f,$50f,$60f,$70f,$80f
        dc.w $90f,$a0f,$b0f,$c0f,$d0f,$e0f,$f0f
        dc.w $f1e,$f2d,$f3c,$f4b,$f5a,$f69,$f78,$f87
        dc.w $f96,$fa5,$fb4,$fc3,$fd2,$fe1,$ff0
        dc.w $ff0,$ff0,$fe0,$fd0,$fc0,$fb0,$fa0,$f90
        dc.w $f80,$f70,$f60,$f50,$f40,$f30,$f20,$f10
        dc.w $f00,$f00,$e01,$d02,$c03,$b04,$a05,$906
        dc.w $807,$708,$609,$50a,$40b,$30c,$20d,$10e,15
        dc.w $0f,$1f,$2f,$3f,$4f,$5f,$6f,$7f,$8f,$9f,$af
        dc.w $bf,$cf,$df,$ef,$ff,$ff,$fe,$fd,$fc,$fb,$fa
        dc.w $f9,$f8,$f7,$f6,$f5,$f4,$f3,$f2,$f1,$f0
        dc.w $1f1,$2f2,$3f3,$4f4,$5f5,$6f6,$7f7,$8f8,$9f9
        dc.w $afa,$bfb,$cfc,$dfd,$efe,$fff,$ffe,$ffd,$ffc,$ffb
        dc.w $ffa,$ff9,$ff8,$ff7,$ff6,$ff5,$ff4,$ff3,$ff2,$ff1,$ff0
        dc.w $fe0,$fd0,$fc0,$fb0,$fa0,$f90,$f80,$f70,$f60,$f50,$f40
        dc.w $f30,$f20,$f10,$f00,$f00,$e00,$d00,$c00,$b00,$a00,$900
        dc.w $800,$700,$600,$500,$400,$300,$200,$100,$0,0
FINECOLORTAB:
```

22.11 Lezione6m

```
; Lezione6m.s        EFFETTO "RIMBALZO" TRAMITE L'USO DI UNA TABELLA


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1        ; Nome lib
        jsr         -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
```

Figura 22.5: Lezione 6l

```
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC

        MOVE.L        #PIC,d0                ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*256,d0        ; + lunghezza bitplane
        addq.w        #8,a1
        dbra        d1,POINTBP

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.w        Rimbalzo        ; Fa "rimbalzare" la PIC tramite una tabella
                                ; gia' predisposta.

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta
```

```
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa volta usiamo una tabella che contiene i valori da sottrarre ai
; puntatori dei bitplanes per simulare un "rimbalzo" della figura, anziche' un
; ovvio movimento SU-GIU con add.l #40 e sub.l #40. Per fare cio' e' bastato
; fare una tabella, appunto, con i valori da sottrarre al puntatore, che
; chiaramente sono multipli del 40, dove 2*40 indica che si saltano 2 linee,
; mentre 3*40 che se ne saltano 3 alla volta:
;
;        dc.l        40,40,2*40,2*40        ; esempio...
;
; Per tornare alla posizione iniziale una volta arrivati in fondo allo schermo
; occorre aggiungere quanto e' stato tolto ai puntatori bitplanes, dunque,
; essendo presente nella routine una sottrazione:
;
;        sub.l        d1,d0        ; sottrai il valore della tabella (d1) all'indirizzo
;                        ; che sta puntando il bplpointer
;
; come facciamo ad AGGIUNGERE con un SUB?? Semplice! Basta SOTTRARRE numeri
; negativi!!! Quanto fa 10-(-1)? Fa 11!!! Dunque nella tabella sono presenti
; i numeri negativi dopo aver toccato "il fondo":
;
;        dc.l        -8*40,-6*40,-5*40                ; risaliamo
;
; un sub.l #-8*40 e' come un add.l #8*40.
; Ricordate pero' che i numeri negativi tengono "il segno" sul bit piu' alto?
; per cui un -40 e' $FFFFFFd8, e' per questo che i valori della tabella sono
; in LONGWORD e non in WORD, per contenere numeri negativi.
; Infatti un:
;
; dc.w -40
;
; Non viene assemblato, da errore, dovete usare .l per i numeri negativi.
;
; Avendo usato i valori .L, bisogna ricordarselo nella routine:
;
; ADDQ.L #4,RIMTABPOINT
; FINERIMBALZTAB-4
; dc.l RIMBALZTAB-4
;
; e non
```

```
;
; ADDQ.L #2,RIMTABPOINT
; FINERIMBALZTAB-2
; dc.l RIMBALZTAB-2
;
; Per quanto riguarda lo spostamento vero e proprio non ci sono novita':
; preleviamo l'indirizzo da BPLPOINTERS, facciamo il SUB con il valore letto
; in tabella e ripuntiamo il nuovo indirizzo.

Rimbalzo:
        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w       2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w       6(a1),d0

        ADDQ.L       #4,RIMTABPOINT        ; Fai puntare alla longword successiva
        MOVE.L       RIMTABPOINT(PC),A0 ; indirizzo contenuto in long RIMTABPOINT
                                ; copiato in a0
        CMP.L       #FINERIMBALZTAB-4,A0 ; Siamo all'ultima longword della TAB?
        BNE.S       NOBSTART2               ; non ancora? allora continua
        MOVE.L       #RIMBALZTAB-4,RIMTABPOINT ; Riparti a puntare dalla prima long
NOBSTART2:
        MOVE.l       (A0),d1                ; copia la long dalla tabella in d1

        sub.l        d1,d0                ; sottraiamo il valore attualmente preso dalla
                                ; tabella, facendo scorrere la figura SU o GIU.

        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w       d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w       d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w       #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP2        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


RIMTABPOINT:                        ; Questa longword "PUNTA" a RIMBALZTAB, ossia
        dc.l        RIMBALZTAB-4        ; contiene l'indirizzo di RIMBALZTAB. Terra'
                                ; l'indirizzo dell'ultima long "letta" dentro
                                ; la tabella. (qua inizia da RIMORTAB-4 in
                                ; quanto Lampeggio inizia con un ADDQ.L #4,C..
                                ; serve per "bilanciare" la prima istruzione.

;        La tabella con i valori "precalcolati" del rimbalzo

RIMBALZTAB:
        dc.l        0,0,0,0,0,0,40,40,40,40,40,40,40,40,40                ; scendiamo
        dc.l        40,40,2*40,2*40
        dc.l        2*40,2*40,2*40,2*40,2*40,2*40
        dc.l        3*40,3*40,3*40,3*40,3*40,4*40,4*40,4*40,5*40,5*40
        dc.l        6*40,8*40                                        ; in fondo
        dc.l        -8*40,-6*40,-5*40                                ; risaliamo
        dc.l        -5*40,-4*40,-4*40,-4*40,-3*40,-3*40,-3*40,-3*40,-3*40
        dc.l        -2*40,-2*40,-2*40,-2*40,-2*40
        dc.l        -2*40,-2*40,-40,-40
        dc.l        -40,-40,-40,-40,-40,-40,-40,-40,-40,0,0,0,0,0        ; siamo in cima
FINERIMBALZTAB:
```

```
        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


        dcb.b        80*40,0        ; spazio azzerato per lo scroll del bitplane

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,

        end
```

Programmare una demo o un gioco significa anche fare una miriade di tabelle.
Quella usata in questo listato potrebbe esser utile per il salto di un ometto
protagonista di un platform; i giochi programmati male e con linguaggi non
adatti spesso difettano piu' del fatto che i movimenti non sono naturali che
della lentezza o altro. Immaginatevi che il protagonista di un platform salti
in alto con un add e improvvisamente arrivato in alto scenda con un sub.
L'effetto sarebbe terribilmente brutto. Anche i movimenti ondeggianti degli
alieni di uno sparatutto contano molto, e sono il frutto di tabelle.
Complicando le cose i programmatori bravucci si fanno un certo numero di
tabelle tutte per il salto del personaggio, e a seconda del movimento di
questo o del tempo di pressione del pulsante fanno fare la curva di salto
giusta secondo la tabella giusta, oppure aggiungono dei valori a quelli della
tabella base, o mischiano i valori di molte tabelle. In casi estremi come
i giochi di flipper bisogna proprio farsi una routine che calcola i rimbalzi e
la gravita', ma questo non esclude che la routine abbia tabelle al suo interno;
comunque in un gioco di flipper si muove solo la pallina (lo schermo del
flipper basta muoverlo cambiando i puntatori dei bitplanes) e si puo' perdere

```
tempo a calcolarsi il movimento, in un altro tipo di gioco si usano le tabelle.
Imparate ad usare la routine che legge i valori dalla tabella e usatela per
modificare esempi precedenti, per far muovere, ad esempio, le barrette copper
in modo strano e oscillante.

Provate a sostituire la tabella con questa: provoca una "fluttuazione"
oscillatoria anziche' un rimbalzo. (usate Amiga+b+c+i)


RIMBALZTAB:
        dc.l        0,0,40,40,40,40,40,40,40,40,40                          ; in cima
        dc.l        40,40,2*40,2*40
        dc.l        2*40,2*40,2*40,2*40,2*40                        ; acceleriamo
        dc.l        3*40,3*40,3*40,3*40,3*40
        dc.l        3*40,3*40,3*40,3*40,3*40
        dc.l        2*40,2*40,2*40,2*40,2*40                        ; deceleriamo
        dc.l        2*40,2*40,40,40
        dc.l        40,40,40,40,40,40,40,40,40,0,0,0,0,0,0,0        ; in fondo
        dc.l        -40,-40,-40,-40,-40,-40,-40,-40,-40
        dc.l        -40,-40,-2*40,-2*40
        dc.l        -2*40,-2*40,-2*40,-2*40,-2*40
        dc.l        -3*40,-3*40,-3*40,-3*40,-3*40,-3*40                         ; acceleriamo
        dc.l        -3*40,-3*40,-3*40,-3*40,-3*40,-3*40
        dc.l        -2*40,-2*40,-2*40,-2*40,-2*40                           ; deceleriamo
        dc.l        -2*40,-2*40,-40,-40
        dc.l        -40,-40,-40,-40,-40,-40,-40,-40,-40,0,0,0,0,0        ; in cima
FINERIMBALZTAB:
```

22.12 Lezione6n

```
; Lezione6n.s        SCROLLING ORIZZONTALE MAGGIORE DI 16 PIXEL, usando il BPLCON1
;                e i BITPLANE POINTERS - TASTO DESTRO PER MUOVERE A SINISTRA

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC

        MOVE.L        #PIC,d0                ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        MOVEQ        #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*256,d0        ; + lunghezza bitplane
        addq.w        #8,a1
        dbra        d1,POINTBP

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
```

```
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s         mouse

        btst         #2,$dff016        ; Tasto destro premuto?
        beq.s         VaiSinistra        ; se si, vai a sinistra!

        bsr.w         Destra                ; Fa avanzare la pic verso destra modificando
                            ; il bplcon1 e i bitplane pointers
        bra.s         Aspetta

VaiSinistra:
        bsr.w         Sinistra        ; Fa indietreggiare la pic verso sinistra.

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001        ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0

OldCop:
        dc.l         0

; Questa routine fa scorrere a destra un bitplane agendo sul BPLCON1 e sui
; puntatori ai bitplanes in copperlist. MIOBPCON1 e' il byte del BPLCON1.

Destra:
        CMP.B         #$ff,MIOBPCON1        ; siamo arrivati al massimo scorrimento? (15)
        BNE.s         CON1ADDA        ; se non ancora, scorri in avanti di 1
                            ; con il BPLCON1

        LEA         BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap         d0                ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w        6(a1),d0

        subq.l        #2,d0                ; punta 16 bit piu' indietro ( la PIC scorre
                            ; verso destra di 16 pixel)
        clr.b         MIOBPCON1        ; azzera lo scroll hardware BPLCON1 ($dff102)
                            ; infatti abbiamo "saltato" 16 pixel con il
                            ; bitplane pointer, ora dobbiamo ricominciare
```

```
                            ; da zero con il $dff102 per scattare a
                            ; destra di un pixel alla volta.

        LEA         BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ         #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0       ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP2       ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts

CON1ADDA:
        add.b         #$11,MIOBPCON1        ; scorri in avanti di 1 pixel
        rts

;        Routine che sposta a sinistra in modo analogo:

Sinistra:
        TST.B         MIOBPCON1        ; siamo arrivati al minimo scorrimento? (00)
        BNE.s         CON1SUBBA        ; se non ancora, scorri indietro di 1
                            ; con il BPLCON1

        LEA         BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w          2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap         d0               ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w        6(a1),d0

        addq.l        #2,d0                ; punta 16 bit piu' avanti ( la PIC scorre
                            ; verso sinistra di 16 pixel)
        move.b        #$FF,MIOBPCON1        ; scroll hardware a 15 - BPLCON1 ($dff102)

        LEA         BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ         #2,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP3:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L         #40*256,d0       ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra         d1,POINTBP3       ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts

CON1SUBBA:
        sub.b         #$11,MIOBPCON1        ; scorri indietro di 1 pixe
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w         $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w         $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w         $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w         $13e,0

        dc.w         $8E,$2c81        ; DiwStrt
        dc.w         $90,$2cc1        ; DiwStop
```

```
        dc.w        $92,$38                 ; DdfStart
        dc.w        $94,$d0                 ; DdfStop
        dc.w        $102                  ; BplCon1
        dc.b        0                ; byte "alto" inutilizzato del $dff102
MIOBPCON1:
        dc.b        0                ; byte "basso" utilizzato del $dff102
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; bits 13 e 12 accesi!! (3 = %011)

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$475        ; color1
        dc.w        $0184,$fff        ; color2
        dc.w        $0186,$ccc        ; color3
        dc.w        $0188,$999        ; color4
        dc.w        $018a,$232        ; color5
        dc.w        $018c,$777        ; color6
        dc.w        $018e,$444        ; color7

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


        dcb.b        80*40,0        ; spazio azzerato per lo scroll del bitplane

PIC:
        incbin        "amiga.320*256*3"        ; qua carichiamo la figura in RAW,

        dcb.b        40,0

        end
```

Orribile l'errore scattoso di visualizzazione al bordo sinistro dello schermo, eh?? Toglierlo non e' difficile, basta cambiare due cosucce, vediamo come e perche': il perche' avviene questo inconveniente e' da ricercare nel fatto che spostando la figura senza informare i canali DMA li troviamo "impreparati" e non fanno in tempo a leggere bene i primi 16 pixel a sinistra.
Per evitare questo cosa possiamo fare? Nulla.
Pero' possiamo far avvenire il pasticcio fuori dallo "schermo visibile", vi ricordate il DIWSTART e il DIWSTOP? Determinano la grandezza della finestra dove sono visualizzati i dati. E' chiaro che se facciamo partire la finestra 16 pixel piu' a destra il problema viene "tappato":

```
        dc.w        $8E,$2c91        ; DiwStrt ($81+16=$91)
```

Provate a cambiare il valore ed eseguite nuovamente il listato. Anche se abbiamo "tappato" l'errore pero' ora abbiamo uno schemo largo 304 pixel anziche' 320 e anche decentrato!!
Ma in nostro aiuto vengono i registri DDFSTART e DDFSTOP! Questi registri si occupano anche loro della grandezza della finestra video, ma in maniera diversa, infatti mentre il DIWSTART/DIWSTOP e' come un cartoncino nero con una fessura ridimensionabile come vediamo nella figura sotto,

```
        ######################
        ######################
```

```
        #####               #####
        #####       figura  #####
        #####               #####
        #####               #####
        #####               #####
        #####               #####
        #####               #####
        ######################
        ######################
```

se cambiamo il DDFSTART/STOP cambiamo proprio la lunghezza di una linea video,
per esempio se allunghiamo lo schermo di 16 pixel, facendolo diventare di
336 pixel per linea, ossia 42 bytes anziche' 40, dovremo visualizzarci una
figura larga proprio 42 pixel per linea.
Il modo OVERSCAN, che allarga la figura visualizzabile oltre i normali 320x256
o 640x256, viene ottenuto con i DDFSTART/STOP, ricordandosi ovviamente di
"allargare" anche la finestra con DIWSTART/DISTOP.
torniamo al problema: noi dobbiamo fare in modo che quell'errore largo 16 pixel
avvenga fuori dalla nostra vista. Basta far cominciare lo schermo col DDFSTART
16 pixel prima, facendoolo finire alla stessa posizione, e lasciare i valori
normali di DIWSTART/DIWSTOP, per cui vediamo sempre 320x256 pixel, ma la
finestra video e' larga in realta' 336 pixel e l'errore sta avvenendo fuori
dalla nostra vista. La figura pero' diventa larga 42 bytes, dunque dobbiamo
bilanciare quei 2 bytes (16 pixel) ogni linea. Come facciamo ogni termine linea
(che ora avviene al pixel 42) a tornare indietro di 2 per visualizzare la
linea correttamente? Insomma per far tornare i conti? Basta sottrarre al modulo
corrente 2. Nel nostro caso, col modulo a ZERO, basta mettere -2.
Per far partire lo schermo 16 pixel prima occorre modificare in questo modo
il DATA FETCH START (DDFSTRT):

```
        dc.w        $92,$30                         ; DDFSTART = $30 (schermo che parte
                                    ; 16 pixel prima, allungandosi a
                                    ; 42 bytes per linea, 336 pixel di
                                    ; larghezza, ma il DIWSTART "nasconde"
                                    ; questi primi 16 pixel con l'errore.


        dc.w        $108,-2                         ; MODULI = -2, dobbiamo "saltare" i
        dc.w        $10a,-2                         ; primi 16 pixel di ogni linea
                                    ; facendoli leggere 2 volte.
```

Fate questa modifica e rimettete a posto il DIWSTART:

```
        dc.w        $8E,$2c81       ; DiwStrt
```

Ora lo scroll e' PERFETTO. C'e' solo il particolare che aumentando tramite
l'OVERSCAN le dimensioni della finestra video viene annullato lo sprite 7,
ossia l'ultimo sprite.

P.S: Se volete dare una sbirciatina all'errore che continua ad esistere in
OVERSCAN fuori dalla finestra fate iniziare il DIWSTART 16 pixel prima:

```
        dc.w        $8E,$2c71       ; DiwStrt
```

E' sempre li'!!!! Ma nessuno lo puo' piu' vedere ora.

Avete visto che era facile togliere l'errore? Basta far iniziare prima di 16
pixel il DDFSTART (a $30) e togliere 2 al valore dei moduli.

22.13 Lezione6o

```
; Lezione6o.s               SCORRIMENTO A DESTRA E SINISTRA DI UN PLAYFIELD PIU'
;                         GRANDE DELLO SCHERMO (qua largo 640 pixel)
;                         Tasto destro per bloccare lo scroll

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

; Attenzione! Per "centrare" l'immagine occorre puntare 2 bytes piu' indietro
; facendo scorrere in avanti la PIC di 16 pixel, infatti la figura comincia
; 16 pixel piu' indietro grazie al DDFSTART (zona "coperta" dove avviene
; l'errore di visualizzazione da nascondere).

        MOVE.L        #BITPLANE-2,d0        ; in d0 mettiamo l'indirizzo del bitplane -2,
                                ; ossia -16 pixel, in quanto i "primi" 16 pixel
                                ; sono "coperti" e dobbiamo "saltarli",
                                ; spostando la PIC in avanti, appunto, di 16
                                ; pixel
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        bsr.w        print                ; Stampa le linee di testo sul playfield
                                ; largo 640 pixel (80 byte per linea)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        btst        #2,$dff016        ; Tasto destro?
        beq.w        Aspetta                ; Se si, non scrollare

        bsr.w        MEGAScrolla        ; Scorrimento orizzontale di una figura larga
                                ; 640 pixel in uno schermo largo 320 pixel.

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
```

```
        jsr        -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; La routine Megascrolla serve solamente ad eseguire 320 volte la routine gia'
; vista "Destra:", dopodiche' eseguire 320 volte la routine "Sinistra:" per
; riportare la figura alla posizione iniziale, di qua il ciclo riparte ecc.
; Per tenere il conto del numero di volte che ha eseguito "Destra:" o
; "Sinistra:" utilizza la word "Contavolte" a cui addiziona "1" ogni FRAME;
; essendo lo schermo video largo 320 pixel e la figura in memoria larga 640,
; per scorrerla occorrera' spostarsi di 320 pixel:
;
; All'inizio:
;         ---------------------------------
;        |                |                |
;        | schermo video |                |
;        | <-   320   -> |                |
;        |                |                |
;        | <- immagine in memoria 640 -> |
;        |                |                |
;        |                |                |
;         ---------------------------------
;
; Quando abbiamo "scrollato" di 320 pixel a Destra:
;         ---------------------------------
;        |                |                |
;        |                 | schermo video |
;        |                | <-   320   -> |
;        |                |                |
;        | <- immagine in memoria 640 -> |
;        |                |                |
;        |                |                |
;         ---------------------------------
;
; Poi altri 320 pixel verso sinistra e torniamo a vedere i primi 320 pixel
; della figura larga 640.
; Tramite il bit 1 della word DestSinFlag viene segnalato se e' necessario
; andare verso destra o verso sinistra. Per scambiare il valore del bit, da
; ZERO ad UNO o da UNO a ZERO e' usata l'istruzione BCHG, ossia BIT CHANGE,
; gia' vista in un'altro listato.

MEGAScrolla:
        addq.w        #1,ContaVolte        ; Segnamo una esecuzione in piu'
        cmp.w        #320,ContaVolte        ; Siamo a 320?
        bne.S        MuoviAncora        ; Se non ancora, sposta ancora
        BCHG.B        #1,DestSinFlag        ; Se siamo a 320, invece, cambia direzione
        CLR.w        ContaVolte        ; di scorrimento e azzera "ContaVolte"
        rts

MuoviAncora:
```

```
        BTST        #1,DestSinFlag        ; Dobbiamo andare a destra o a sinistra?
        BEQ.S        VaiSinistra
        bsr.s        Destra                ; Scorri un pixel verso destra
        rts

VaiSinistra:
        bsr.s        Sinistra        ; Scorri un pixel verso sinistra
        rts

; Questa word conta quante volte ci siamo spostati a Destra o a Sinistra

ContaVolte:
        DC.W        0

; Quando il bit 1 di DestSinFlag e' a ZERO la routine scorre a Sinistra, se
; e' ad 1 scorre a Destra.

DestSinFlag:
        DC.W        0

; Questa routine fa scorrere a destra un bitplane agendo sul BPLCON1 e sui
; puntatori ai bitplanes in copperlist. MIOBPCON1 e' il byte del BPLCON1.

Destra:
        CMP.B        #$ff,MIOBPCON1        ; siamo arrivati al massimo scorrimento? (15)
        BNE.s        CON1ADDA        ; se non ancora, scorri in avanti di 1
                                ; con il BPLCON1

        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w        6(a1),d0

        subq.l        #2,d0                ; punta 16 bit piu' indietro ( la PIC scorre
                                ; verso destra di 16 pixel)
        clr.b        MIOBPCON1        ; azzera lo scroll hardware BPLCON1 ($dff102)
                                ; infatti abbiamo "saltato" 16 pixel con il
                                ; bitplane pointer, ora dobbiamo ricominciare
                                ; da zero con il $dff102 per scattare a
                                ; destra di un pixel alla volta.

        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        rts

CON1ADDA:
        add.b        #$11,MIOBPCON1        ; scorri in avanti di 1 pixel
        rts

;        Routine che sposta a sinistra in modo analogo:

Sinistra:
        TST.B        MIOBPCON1        ; siamo arrivati al minimo scorrimento? (00)
        BNE.s        CON1SUBBA        ; se non ancora, scorri indietro di 1
                                ; con il BPLCON1

        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w        6(a1),d0
```

```
        addq.l        #2,d0                ; punta 16 bit piu' avanti ( la PIC scorre
                                  ; verso sinistra di 16 pixel)
        move.b        #$FF,MIOBPCON1        ; scroll hardware a 15 - BPLCON1 ($dff102)

        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap         d0               ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        rts

CON1SUBBA:
        sub.b        #$11,MIOBPCON1        ; scorri indietro di 1 pixe
        rts



;        Routine che stampa caratteri larghi 8x8 pixel (su schermo HIRES)

PRINT:
        LEA        TESTO(PC),A0        ; Indirizzo del testo da stampare in a0
        LEA        BITPLANE,A3        ; Indirizzo del bitplane destinazione in a3
        MOVEQ        #25-1,D3        ; NUMERO RIGHE DA STAMPARE: 25
PRINTRIGA:
        MOVEQ        #80-1,D0        ; NUMERO COLONNE PER RIGA: 80 (hires!)
PRINTCHAR2:
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                  ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                  ; DELLO SPAZIO (che e' $20), in $00, quello
                                  ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                  ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

                                  ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,80(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,80*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,80*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,80*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,80*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,80*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,80*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.w        #1,A3                ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA        D0,PRINTCHAR2        ; STAMPIAMO D0 (80) CARATTERI PER RIGA

        ADD.W        #80*7,A3        ; ANDIAMO A CAPO

        DBRA        D3,PRINTRIGA        ; FACCIAMO D3 RIGHE

        RTS


                ; numero caratteri per linea: 80, dunque 2 di queste da 40!
TESTO:             ;                 1111111111222222222333333333334
             ;         12345678901234567890123456789012345678901234567890
        dc.b        '   PRIMA RIGA  IN HIRES 640 PIXEL DI LAR' ; 1a \ PRIMA RIGA
        dc.b        'GHEZZA!  -- -- --   SEMPRE LA PRIMA RIGA' ; 1b /
        dc.b        '               SECONDA RIGA            ' ; 2  \ SECONDA RIGA
```

```
        dc.b        'ANCORA SECONDA RIGA                    ' ;    /
        dc.b        '     /\  /                             ' ; 3
        dc.b        '                                       ' ;
        dc.b        '    /  \/                              ' ; 4
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 5
        dc.b        '                                       ' ;
        dc.b        '        SESTA RIGA                     ' ; 6
        dc.b        '                         FINE SESTA RIGA ' ;
        dc.b        '                                       ' ; 7
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 8
        dc.b        '                                       ' ;
        dc.b        'FABIO CIUCCI COMMUNICATION INTERNATIONAL' ; 9
        dc.b        ' MARKETING TRUST TRADEMARK COPYRIGHTED  ' ;
        dc.b        '                                       ' ; 10
        dc.b        '                                       ' ;
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}     ' ; 11
        dc.b        '   PROVE TECNICHE DI TRASMISSIONE      ' ;
        dc.b        '                                       ' ; 12
        dc.b        '                                       ' ;
        dc.b        '    LA PALINGENETICA OBLITERAZIONE DELL' ; 13
        dc.b        "'IO TRASCENDENTALE CHE SI IMMEDESIMA    " ;
        dc.b        '                                       ' ; 14
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 15
        dc.b        '                                       ' ;
        dc.b        '  Nel mezzo del cammin di nostra vita  ' ; 16
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 17
        dc.b        '                                       ' ;
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa   ' ; 18
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 19
        dc.b        '                                       ' ;
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA    ' ; 20
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 21
        dc.b        '                                       ' ;
        dc.b        '  AHI Quanto a DIR QUAL ERA...         ' ; 22
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 23
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 24
        dc.b        '                                       ' ;
        dc.b        ' C:\>_                                 ' ; 25
        dc.b        '                                       ' ;
        dc.b        '                                       ' ; 26
        dc.b        '                                       ' ;

        EVEN


        SECTION       GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0
```

```
        dc.w        $8e,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$30               ; DdfStart (modificato per SCROLL)
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102                ; BplCon1
        dc.b        0                ; byte "alto" inutilizzato del $dff102
MIOBPCON1:
        dc.b        0                ; byte "basso" utilizzato del $dff102
        dc.w        $104,0              ; BplCon2
        dc.w        $108,40-2        ; Bpl1Mod (40 per la figura larga 640, il -2
        dc.w        $10a,40-2        ; Bpl2Mod (per bilanciare il DDFSTART

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 - 1 bitplane LOWRES

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo        bitplane

        dc.w        $180,$103        ; color0 - SFONDO
        dc.w        $182,$4ff        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
;        incbin        "metal.fnt"
;        incbin        "normal.fnt"
        incbin        "nice.fnt"


        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        80*256        ; un bitplane largo 640x256 (come l'HIRES)

        end
```

In questo listato l'unica vera novita' sta nel fatto che scorriamo una figura piu' grande dello schermo anziche' una grande 320 pixel.
Innanzitutto quando lo schermo e' in LOWRES con valori DDFSTART/STOP normali il modulo "automatico" e' 40, ossia l'immagine viene considerata fatta di linee di 40 bytes. Se invece abbiamo in memoria una figura larga 640 pixel, come in questo caso, dobbiamo cambiare il modulo. Infatti il fatto che la figura in memoria e' piu' grande non interessa al Copper, il quale visualizza come sempre uno schermo in LOWRES con modulo 40. Noi possiamo pero' cambiare il modulo tramite i registri BPL1MOD e BPL2MOD: il modulo viene aggiunto al modulo corrente, che e' 40, dunque bastera' un:

```
        dc.w        $108,40               ; Bpl1Mod (40 per la figura larga 640)
        dc.w        $10a,40               ; Bpl2Mod
```

Per far "saltare" alla fine di ogni linea di 320 pixel (40 bytes) i 40 bytes che sono "fuori video", facendo continuare la visualizzazione dall'inizio della linea seguente:

```
         40 bytes          40 bytes (saltati ogni volta col modulo = 40)
         --------------------------------
        |                |                |
        | schermo video |                |
        | <-   320   -> |                |
        |                |                |
```

```
        | <- immagine in memoria 640 -> |
        |                |               |
        |                |               |
         -------------------------------
```

Ora, avendo visualizzato la parte Destra della figura larga 640 pixel su uno
schermo largo 320 SEMPLICEMENTE mettendo i moduli a 40, dobbiamo fare la
stessa modifica dell'esempio Lezione6n.s per "nascondere" i primi 16 pixel
dove avviene l'errore di visualizzazione a cusa dello scroll.
Dobbiamo allora far partire lo schermo 16 pixel prima modificando il DDFSTART:

```
        dc.w        $92,$30                         ; DDFSTART = $30 (schermo che parte
                                        ; 16 pixel prima, allungandosi a
                                        ; 42 bytes per linea, 336 pixel di
                                        ; larghezza, ma il DIWSTART "nasconde"
                                        ; questi primi 16 pixel con l'errore.
```

E, in quanto abbiamo allargato lo schermo facendolo andare "a capo" ogni 42
bytes anziche' 40, e' necessario bilanciare sottraendo 2 ai moduli, che
nel nostro caso erano a 40, e andranno a 38:

```
        dc.w        $108,40-2       ; Bpl1Mod (40 per la figura larga 640, il -2
        dc.w        $10a,40-2       ; Bpl2Mod (per bilanciare il DDFSTART
```

In fondo non si puo' dire che uno scroll di questo tipo e' "difficile", l'unica
difficolta' sta nel ricordarsi di sistemare bene MODULI/DDFSTART/INDIRIZZO DEL
BITPLANE. Infatti c'e' anche un'altra "novita'" rispetto a Lezione6n.s:

```
; Attenzione! Per "centrare" l'immagine occorre puntare 2 bytes piu' indietro
; facendo scorrere in avanti la PIC di 16 pixel, infatti la figura comincia
; 16 pixel piu' indietro grazie al DDFSTART (zona "coperta" dove avviene
; l'errore di visualizzazione da nascondere).

        MOVE.L      #BITPLANE-2,d0      ; in d0 mettiamo l'indirizzo del bitplane -2,
                            ; ossia -16 pixel, in quanto i "primi" 16 pixel
                            ; sono "coperti" e dobbiamo "saltarli",
                            ; spostando la PIC in avanti, appunto, di 16
                            ; pixel
```

Infatti abbiamo "nascosto" i primi 16 pixel, dunque nasconderemmo i primi 2
caratteri del testo, (8pixel ciascuno*2=16 pixel). Invece "spostando" la figura
in avanti di 16 pixel vediamo correttamente anche i primi 16 pixel e la figura
appare centrata, non spostata verso sinistra di 16 pixel come in Lezione6n.s.
Provate a togliere il -2 dal "MOVE.L #BITPLANE-2,d0" e mettete un ; alla
routine

```
;        bsr.w      MEGAScrolla
```

in modo da avere una figura FERMA e noterete che i primi 16 pixel mancano, e
ce ne sono 2 in piu' a destra, ossia la figura parte 16 pixel prima del normale
Per verificare cio', "scopriamo" i primi 16 pixel:

```
        dc.w        $8e,$2c71       ; DiwStrt ($81-16=$71)
```

Ecco li' i primi 16 pixel "spariti!". Rimettete a posto il -2 e togliete il ;
dalla routine lasciando i primi 16 pixel "scoperti" e vedrete come l'errore
fatidico dello scroll avviene in sordina "dietro le quinte".

Figura 22.6: Lezione 6o

22.14 Lezione6p

```
; Lezione6p.s        STAMPIAMO LO SCHERMO UN CARATTERE OGNI FOTOGRAMMA

        SECTION        CiriCop,CODE

Inizio:
        move.l         4.w,a6                ; Execbase
        jsr           -$78(a6)         ; Disable
        lea           GfxName(PC),a1         ; Nome lib
        jsr           -$198(a6)         ; OpenLibrary
        move.l         d0,GfxBase
        move.l         d0,a6
        move.l         $26(a6),OldCop        ; salviamo la vecchia COP

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L         #BITPLANE,d0
        LEA           BPLPOINTERS,A1
        move.w         d0,6(a1)
        swap          d0
        move.w         d0,2(a1)

        move.l         #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w         d0,$dff088                ; Facciamo partire la COP
        move.w         #0,$dff1fc                ; Disattiva l'AGA
        move.w         #$c00,$dff106                ; Disattiva l'AGA

mouse:
        cmpi.b         #$ff,$dff006        ; Linea 255?
        bne.s          mouse
```

```
        bsr.s        PrintCarattere        ; Stampa un carattere alla volta

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


;        Questa routine e' una specie di ibrido tra la routine normale del
;        PRINT e quella della TABELLA, infatti utiliziamo il TESTO in maniera
;        analoga ad una tabella, prendendone un valore solamente ogni FOTOGRAMMA
;        e stampandolo. D'altronde dobbiamo salvare anche l'indirizzo nel
;        bitplane dell'ultima posizione raggiunta nel print, per stampare il
;        carattere seguente dopo quello precedente. Per mantenere tra un frame
;        e l'altro l'indirizzo del carattere nel testo e del punto del bitplane
;        dove siamo arrivati sono usate 2 longword PUNTATORI:
;
; PuntaTesto:
;        dc.l        TESTO
;
; PuntaBitplane:
;        dc.l        BITPLANE
;
;        Ogni volta che la routine viene eseguita viene stampato un solo
;        carattere e viene aggiornato sia il puntatore del TESTO, con un
;        ADDQ.L #1 che lo sposta al carattere dopo (essendo un carattere
;        lungo un byte), sia il puntatore del BITPLANE, infatti ogni carattere
;        ha il suo posto nel bitplane.
;        Il primo problema e' che ogni 40 caratteri bisogna "ANDARE A CAPO",
;        ossia aggiungere 40*7 al puntatore del BITPLANE. Per risolvere questo
;        e' bastato aggiungere uno ZERO alla fine di ogni riga di testo e
;        delle istruzioni che controllano se il byte da stampare e' zero: se
;        si tratta dello zero allora significa che siamo alla fine della riga,
;        dunque viene aggiunto 40*7 al puntatore bitplane ed 1 a quello testo
;        per saltare lo zero e puntare al primo carattere della riga dopo.
;        Il secondo problema e' che una volta raggiunta la fine del testo
;        bisogna smettere di stampare caratteri. Per convenzione terminando
;        la riga con $FF anziche' con $00 indichiamo la fine del testo, basta
;        controllare se il byte da leggere e' $FF e uscire senza stampare e
```

```
;        senza far avanzare il puntatore PUNTATESTO, per cui ogni volta che
;        viene eseguito PRINTcarattere dopo aver stampato l'intero testo usciamo
;        senza compiere operazioni, in quanto il carattere e' sempre $FF.
;        NOTA: potete "inventare" vari numeri "speciali" da inserire nel
;        testo per varie funzioni, basta che non siano numeri compresi tra $20
;        e $80, ossia tra i byte dedicati ai caratteri.

PRINTcarattere:
        MOVE.L        PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        CMP.B        #$ff,d2                ; Segnale di fine testo? ($FF)
        beq.s        FineTesto        ; Se si, esci senza stampare
        TST.B        d2                ; Segnale di fine riga? ($00)
        bne.s        NonFineRiga        ; Se no, non andare a capo

        ADD.L        #40*7,PuntaBITPLANE        ; ANDIAMO A CAPO
        ADDQ.L        #1,PuntaTesto                ; primo carattere riga dopo
                                        ; (saltiamo lo ZERO)
        move.b        (a0)+,d2                ; primo carattere della riga dopo
                                        ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        MULU.W        #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L        PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B        (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.L        #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L        #1,PuntaTesto        ; prossimo carattere da stampare

FineTesto:
        RTS


PuntaTesto:
        dc.l        TESTO

PuntaBitplane:
        dc.l        BITPLANE

;        $00 per "fine linea" - $FF per "fine testo"

                ; numero caratteri per linea: 40
TESTO:             ;                1111111111222222222333333333334
        dc.b        '   PRIMA RIGA                          ',0 ; 1
```

```
        dc.b        '               SECONDA RIGA            ',0 ; 2
        dc.b        '     /\  /                             ',0 ; 3
        dc.b        '    /  \/                              ',0 ; 4
        dc.b        '                                       ',0 ; 5
        dc.b        '        SESTA RIGA                     ',0 ; 6
        dc.b        '                                       ',0 ; 7
        dc.b        '                                       ',0 ; 8
        dc.b        'FABIO CIUCCI COMMUNICATION INTERNATIONAL',0 ; 9
        dc.b        '                                       ',0 ; 10
        dc.b        '   1234567890 !@#$%^&*()_+|\=-[]{}     ',0 ; 11
        dc.b        '                                       ',0 ; 12
        dc.b        '     LA PALINGENETICA OBLITERAZIONE    ',0 ; 15
        dc.b        '                                       ',0 ; 16
        dc.b        '                                       ',0 ; 17
        dc.b        '  Nel mezzo del cammin di nostra vita  ',0 ; 18
        dc.b        '                                       ',0 ; 19
        dc.b        '    Mi RitRoVaI pEr UnA sELva oScuRa   ',0 ; 20
        dc.b        '                                       ',0 ; 21
        dc.b        '    CHE LA DIRITTA VIA ERA SMARRITA    ',0 ; 22
        dc.b        '                                       ',0 ; 23
        dc.b        '  AHI Quanto a DIR QUAL ERA...         ',$FF ; 24 FINE


        EVEN



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"
;        incbin        "normal.fnt"
;        incbin        "nice.fnt"
```

```
        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

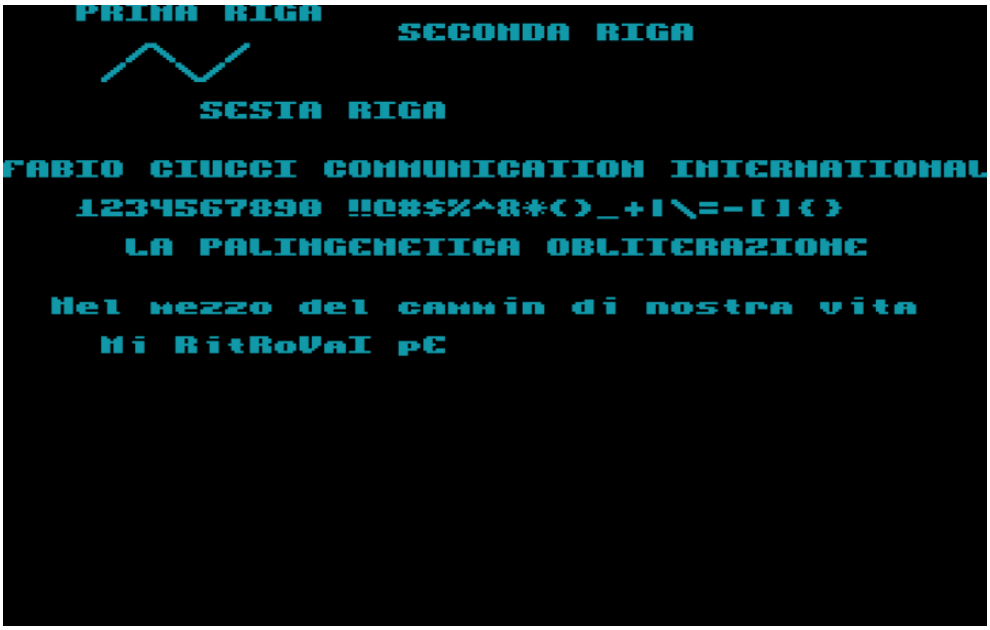


Figura 22.7: Lezione 6p

22.15 Lezione6q

```
; Lezione6q.s        Alcune "scacchiere" screate sullo schermo
;                - ALTERNARE TASTO SINISTRO-DESTRO-SINISTRO del mouse per
;                vedere le scacchiere ed uscire

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
```

```
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w        d0,$dff088                ; Facciamo partire la COP
        move.w        #0,$dff1fc                ; Disattiva l'AGA
        move.w        #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        GRIGLIA1

mouse:
        btst        #6,$bfe001        ; tasto sinistro?
        bne.s        mouse

        bsr.w        GRIGLIA2

mouse2:
        btst        #2,$dff016        ; tasto destro?
        bne.s        Mouse2

        bsr.w        GRIGLIA3

mouse3:
        btst        #6,$bfe001        ; tasto sinistro?
        bne.s        mouse3

        bsr.w        GRIGLIA4

mouse4:
        btst        #2,$dff016        ; tasto destro?
        bne.s        Mouse4


        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


; questa routine fa una scacchiera con quadrati di 8 pixel di lato

GRIGLIA1:
        LEA        BITPLANE,a0        ; Indirizzo bitplane destinazione

        MOVEQ        #16-1,d0        ; 16 coppie di quadretti alti 8 pixel
                                ; 16*2*8=256 riempimento completo dello schermo
```

```
FaiCoppia:
        move.l        #(20*8)-1,d1        ; 20 words per riempire 1 linea
                                  ; 8 linee da riempire
FaiUNO:
        move.w        #%1111111100000000,(a0)+ ; lunghezza quadretto ad 1 = 8 pixel
                                       ; quadretto azzerato = 8 pixel
        dbra        d1,FaiUNO                ; fai 8 linee #.#.#.#.#.#.#.#.#

        move.l        #(20*8)-1,d1        ; 20 words per riempire 1 linea
                                  ; 8 linee da riempire
FaiALTRO:
        move.w        #%0000000011111111,(a0)+ ; lunghezza quadretto azzerato = 8
                                       ; quadretto ad 1 = 8 pixel
        dbra        d1,FaiAltro                ; fai 8 linee .#.#.#.#.#.#.#.#.

        DBRA        d0,FaiCoppia                ; fai 16 coppie di quadretti
                                      ; #.#.#.#.#.#.#.#.#
        rts                              ; .#.#.#.#.#.#.#.#.

; questa routine fa una scacchiera con quadrati di 4 pixel di lato

GRIGLIA2:
        LEA        BITPLANE,a0        ; Indirizzo bitplane destinazione

        MOVEQ        #32-1,d0        ; 32 coppie di quadretti alti 4 pixel
                                  ; 32*2*4=256 riempimento completo dello schermo
FaiCoppia2:
        move.l        #(40*4)-1,d1        ; 40 bytes per riempire 1 linea
                                  ; 4 linee da riempire
FaiUNO2:
        move.b        #%11110000,(a0)+         ; lunghezza quadretto ad 1 = 4 pixel
                                      ; quadretto azzerato = 4 pixel
        dbra        d1,FaiUNO2                ; fai 4 linee #.#.#.#.#.#.#.#.#

        move.l        #(40*4)-1,d1        ; 40 bytes per riempire 1 linea
                                  ; 4 linee da riempire
FaiALTRO2:
        move.b        #%00001111,(a0)+         ; lunghezza quadretto azzerato=4 pixel
                                      ; quadretto ad 1 = 4 pixel
        dbra        d1,FaiAltro2                ; fai 8 linee .#.#.#.#.#.#.#.#.

        DBRA        d0,FaiCoppia2                ; fai 32 coppie di quadretti
                                      ; #.#.#.#.#.#.#.#.#
        rts                              ; .#.#.#.#.#.#.#.#.

; questa routine fa una scacchiera con quadrati di 16 pixel di lato

GRIGLIA3:
        LEA        BITPLANE,a0        ; Indirizzo bitplane destinazione

        MOVEQ        #8-1,d0                ; 8 coppie di quadretti alti 16 pixel
                                  ; 8*2*16=256 riempimento completo dello schermo
FaiCoppia3:
        move.l        #(10*16)-1,d1        ; 10 lingwords per riempire 1 linea
                                  ; 16 linee da riempire
FaiUNO3:
        move.l        #%11111111111111110000000000000000,(a0)+
                                      ; lunghezza quadretto ad 1 = 16 pixel
                                      ; quadretto azzerato = 16 pixel
        dbra        d1,FaiUNO3                ; fai 16 linee #.#.#.#.#.#.#.#.#

        move.l        #(10*16)-1,d1        ; 10 lingwords per riempire 1 linea
```

```
                            ; 16 linee da riempire
FaiALTRO3:
        move.l        #%00000000000000001111111111111111,(a0)+
                                    ; lunghezza quadretto azzerato = 16
                                    ; quadretto ad 1 = 16 pixel
        dbra        d1,FaiAltro3                ; fai 8 linee .#.#.#.#.#.#.#.#.

        DBRA        d0,FaiCoppia3                ; fai 8 coppie di quadretti
                                     ; #.#.#.#.#.#.#.#.#.#
        rts                                ; .#.#.#.#.#.#.#.#.

        ; griglia "fantasia"

GRIGLIA4:
        LEA        BITPLANE,a0        ; Indirizzo bitplane destinazione

        MOVEQ        #8-1,d0                ; 8 coppie di quadretti alti 16 pixel
                            ; 8*2*16=256 riempimento completo dello schermo
FaiCoppia4:
        move.l        #(10*16)-1,d1        ; 10 lingwords per riempire 1 linea
                            ; 16 linee da riempire
FaiUNO4:
        move.l        #%11111000000000001111100000000000,(a0)+
                                    ; lunghezza quadretto ad 1 = 4 pixel
                                    ; quadretto azzerato = 12 pixel
        dbra        d1,FaiUNO4                ; fai 16 linee #.#.#.#.#.#.#.#.#.#

        move.l        #(10*16)-1,d1        ; 10 lingwords per riempire 1 linea
                            ; 16 linee da riempire
FaiALTRO4:
        move.l        #%00000000000011111000000000011111,(a0)+
                                    ; lunghezza quadretto azzerato = 12
                                    ; quadretto ad 1 = 4 pixel
        dbra        d1,FaiAltro4                ; fai 8 linee .#.#.#.#.#.#.#.#.#.

        DBRA        d0,FaiCoppia4                ; fai 8 coppie di quadretti
                                     ; #.#.#.#.#.#.#.#.#.#
        rts                                ; .#.#.#.#.#.#.#.#.#.

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
```

```
        dc.w        $0182,$19a        ; color1 - DISEGNO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Se avete bisogno di uno sfondo geometrico o ripetitivo potete farvelo con una routine anziche' disegnarlo, risparmiando lo spazio della figura. Questo e' solo un esempio di quello che si puo' fare, si possono anche ripetere piccoli disegni sullo schermo copiandoli uno accanto all'altro come mattoncini.

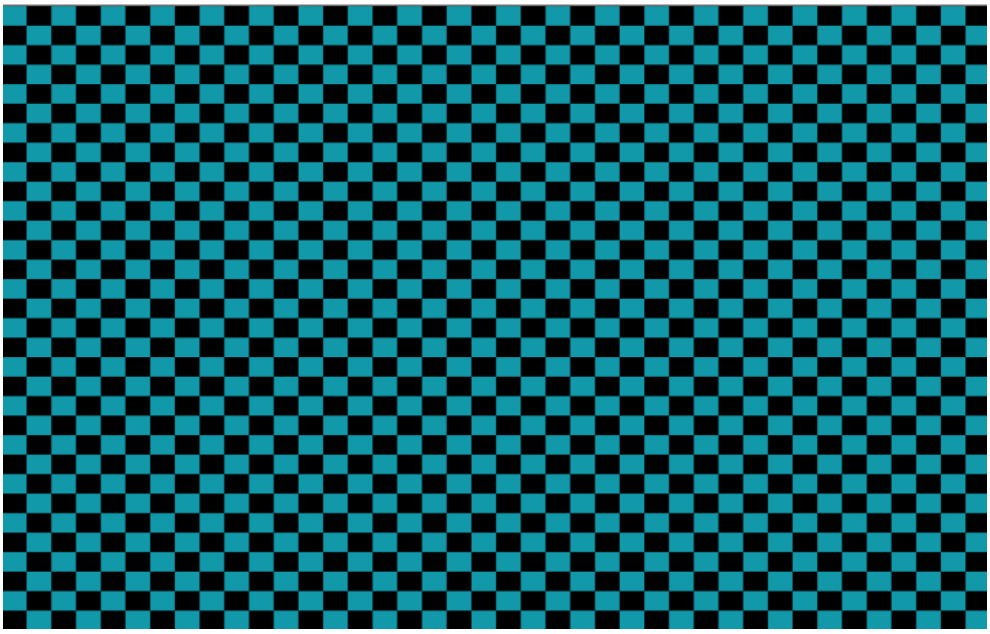


Figura 22.8: Lezione 6q

22.16 Lezione6r

```
; Lezione6r.s        RIEPILOGO DELLA LEZIONE 6 - VARIE ROUTINES DELLA LEZIONE
;                COMBINATE INSIEME + ROUTINE MUSICALE

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
```

```
        jsr        -$198(a6)        ; OpenLibrary
        move.l       d0,GfxBase
        move.l       d0,a6
        move.l       $26(a6),OldCop        ; salviamo la vecchia COP

;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L       #BITPLANETESTO-2,d0
        LEA        BPLPOINTERS,A1
        move.w       d0,6(a1)
        swap        d0
        move.w       d0,2(a1)

        MOVE.L       #BITPLANEGRIGLIA-2,d0
        LEA        BPLPOINTERS2,A1
        move.w       d0,6(a1)
        swap        d0
        move.w       d0,2(a1)

        move.l       #COPPERLIST,$dff080        ; Puntiamo la nostra COP
        move.w       d0,$dff088                ; Facciamo partire la COP
        move.w       #0,$dff1fc                ; Disattiva l'AGA
        move.w       #$c00,$dff106                ; Disattiva l'AGA

        bsr.w        griglia3        ; Fai la scacchiera su BITPLANEGRIGLIA

        bsr.w        mt_init                ; Inizializza routine musicale

mouse:
        cmpi.b       #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        PrintCarattere        ; Stampa un carattere alla volta
        bsr.w        MEGAScrolla        ; Esegue lo scroll dello schermo largo 640
                            ; pixel su uno di 320
        bsr.w        Rimbalzo        ; Fa rimbalzare il bitplane TESTO
        bsr.w        mt_music        ; Suona la musica

Aspetta:
        cmpi.b       #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        bsr.w        mt_end                ; Termina la routine musicale

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
```

```
        dc.l        0

OldCop:
        dc.l        0

;***************************************************************************
;*        Stampa un carattere alla volta su schermo largo 640 pixel        *
;***************************************************************************

PRINTcarattere:
        MOVE.L       PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ       #0,D2                ; Pulisci d2
        MOVE.B       (A0)+,D2        ; Prossimo carattere in d2
        CMP.B       #$ff,d2               ; Segnale di fine testo? ($FF)
        beq.s       FineTesto        ; Se si, esci senza stampare
        TST.B       d2                ; Segnale di fine riga? ($00)
        bne.s       NonFineRiga        ; Se no, non andare a capo

        ADD.L       #80*7,PuntaBITPLANE        ; ANDIAMO A CAPO
        ADDQ.L       #1,PuntaTesto                ; primo carattere riga dopo
                                    ; (saltiamo lo ZERO)
        move.b       (a0)+,d2                ; primo carattere della riga dopo
                                    ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B       #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                            ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                            ; DELLO SPAZIO (che e' $20), in $00, quello
                            ; DELL'ASTERISCO ($21), in $01...
        MULU.W       #8,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                            ; essendo i caratteri alti 8 pixel
        MOVE.L       D2,A2
        ADD.L       #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L       PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                            ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B       (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B       (A2)+,80(A3)        ; stampa LA LINEA 2  " "
        MOVE.B       (A2)+,80*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B       (A2)+,80*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B       (A2)+,80*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B       (A2)+,80*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B       (A2)+,80*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B       (A2)+,80*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.L       #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L       #1,PuntaTesto        ; prossimo carattere da stampare

FineTesto:
        RTS


PuntaTesto:
        dc.l        TESTO

PuntaBitplane:
        dc.l        BITPLANETESTO

;        $00 per "fine linea" - $FF per "fine testo"

               ; numero caratteri per linea: 80
```

```
TESTO:
        dc.b        "          QUESTA DEMO RIASSUME LA LEZION"
        dc.b        "E 6 IN QUANTO CONTIENE SIA LA           ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          ROUTINE DI STAMPA DEI CARATTER"
        dc.b        "I DI 8x8 PIXEL, SIA LO SCROLL           ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          ORIZZONTALE TRAMITE IL BPLCON1"
        dc.b        " ($dff102) E I BPLPOINTERS, SIA         ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          L'USO DI UNA TABELLA DI VALORI"
        dc.b        " PREDEFINITI PER IL MOVIMENTO           ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          OSCILLATORIO VERTICALE DI QUES"
        dc.b        "TO TESTO.                               ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          IL PLAYFIELD DOVE VIENE STAMPA"
        dc.b        "TO QUESTO TESTO HA LE DIMENSIONI        ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          DI UNO SCHERMO HIRES, OSSIA 64"
        dc.b        "0 PIXEL DI LARGHEZZA PER 256 DI         ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          ALTEZZA, E VIENE SPOSTATO SIA "
        dc.b        "ORIZZONTALMENTE, SIA VERTICALMENTE,     ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          MENTRE IL BITPLANE CHE CONTIEN"
        dc.b        "E LA GRIGLIA VIENE SPOSTATO SOLO        ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          ORIZZONTALMENTE. LO SCROLL VER"
        dc.b        "TICALE, ESSENDO DETERMINATO DA UNA      ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          TABELLA, HA VELOCITA' VARIABIL"
        dc.b        "E, MENTRE LO SCROLL ORIZZONTALE E'      ",0
        dc.b        "                                        "
        dc.b        "                                        ",0
        dc.b        "          SEMPRE DI 2 PIXEL PER FOTOGRAM"
        dc.b        "MA.                                     ",$FF


        EVEN

;***************************************************************************
;*   Esegue uno scroll di 320 pixel a destra e sinistra (Lezione6o.s)       *
;***************************************************************************

; NOTA: Modificata in modo da agire anche sul bitplane della GRIGLIA

MEGAScrolla:
        addq.w       #1,ContaVolte        ; Segnamo una esecuzione in piu'
        cmp.w       #160,ContaVolte        ; Siamo a 160? Allora abbiamo scrollato 320
                                ; pixel, dato che eseguiamo 2 volte la routine
```

```
                            ; DESTRA o SINISTRA ogni FRAME per andare al
                            ; doppio della velocita'
        bne.S       MuoviAncora        ; Se non ancora, sposta ancora
        BCHG.B       #1,DestSinFlag        ; Se siamo a 160, invece, cambia direzione
        CLR.w       ContaVolte        ; di scorrimento e azzera "ContaVolte"
        rts

MuoviAncora:
        BTST        #1,DestSinFlag        ; Dobbiamo andare a destra o a sinistra?
        BEQ.S        VaiSinistra
        bsr.s       Destra                ; Scorri un pixel verso destra
        bsr.s       Destra                ; Scorri un pixel verso destra
                            ; (2 pixel per frame, dunque doppia velocita')
        rts

VaiSinistra:
        bsr.s       Sinistra        ; Scorri un pixel verso sinistra
        bsr.s       Sinistra        ; Scorri un pixel verso sinistra
                            ; (2 pixel per frame, dunque doppia velocita')
        rts

; Questa word conta quante volte ci siamo spostati a Destra o a Sinistra

ContaVolte:
        DC.W        0

; Quando il bit 1 di DestSinFlag e' a ZERO la routine scorre a Sinistra, se
; e' ad 1 scorre a Destra.

DestSinFlag:
        DC.W        0

; Questa routine fa scorrere a destra un bitplane agendo sul BPLCON1 e sui
; puntatori ai bitplanes in copperlist. MIOBPCON1 e' il byte del BPLCON1.

Destra:
        CMP.B       #$ff,MIOBPCON1        ; siamo arrivati al massimo scorrimento? (15)
        BNE.s       CON1ADDA        ; se non ancora, scorri in avanti di 1
                            ; con il BPLCON1

        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w       2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w       6(a1),d0

        LEA        BPLPOINTERS2,A2        ; Con queste 4 istruzioni preleviamo dalla
        move.w       2(a2),d1        ; copperlist l'indirizzo dove sta puntando
        swap        d1                ; attualmente il $dff0e4 e lo poiniamo in d0
        move.w       6(a2),d1

        subq.l       #2,d0                ; punta 16 bit piu' indietro ( la PIC scorre
                            ; verso destra di 16 pixel) - TESTO

        subq.l       #2,d1                ; punta 16 bit piu' indietro ( la PIC scorre
                            ; verso destra di 16 pixel) - GRIGLIA

        clr.b       MIOBPCON1        ; azzera lo scroll hardware BPLCON1 ($dff102)
                            ; infatti abbiamo "saltato" 16 pixel con il
                            ; bitplane pointer, ora dobbiamo ricominciare
                            ; da zero con il $dff102 per scattare a
                            ; destra di un pixel alla volta.
```

```
;        Puntiamo il bitplane TESTO

        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

;        Puntiamo il bitplane GRIGLIA

        move.w        d1,6(a2)        ; copia la word BASSA dell'indirizzo del plane
        swap        d1                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d1,2(a2)        ; copia la word ALTA dell'indirizzo del plane

        rts

CON1ADDA:
        add.b        #$11,MIOBPCON1        ; scorri in avanti di 1 pixel
        rts

;        Routine che sposta a sinistra in modo analogo:

Sinistra:
        TST.B        MIOBPCON1        ; siamo arrivati al minimo scorrimento? (00)
        BNE.s        CON1SUBBA        ; se non ancora, scorri indietro di 1
                                ; con il BPLCON1

        LEA        BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w        6(a1),d0

        LEA        BPLPOINTERS2,A2        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a2),d1        ; copperlist l'indirizzo dove sta puntando
        swap        d1                ; attualmente il $dff0e4 e lo poiniamo in d0
        move.w        6(a2),d1

        addq.l        #2,d0                ; punta 16 bit piu' avanti ( la PIC scorre
                                ; verso sinistra di 16 pixel) - TESTO

        addq.l        #2,d1                ; punta 16 bit piu' avanti ( la PIC scorre
                                ; verso sinistra di 16 pixel) - GRIGLIA

        move.b        #$FF,MIOBPCON1        ; scroll hardware a 15 - BPLCON1 ($dff102)

;        Puntiamo il bitplane TESTO

        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

;        Puntiamo il bitplane GRIGLIA

        move.w        d1,6(a2)        ; copia la word BASSA dell'indirizzo del plane
        swap        d1                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d1,2(a2)        ; copia la word ALTA dell'indirizzo del plane

        rts

CON1SUBBA:
        sub.b        #$11,MIOBPCON1        ; scorri indietro di 1 pixel
        rts
```

```
;*********************************************************************
;*   Fa oscillare SU/GIU tramite l'uso di una tabella (Lezione6m.s)        *
;*********************************************************************

Rimbalzo:
        LEA         BPLPOINTERS,A1        ; Con queste 4 istruzioni preleviamo dalla
        move.w        2(a1),d0          ; copperlist l'indirizzo dove sta puntando
        swap         d0                 ; attualmente il $dff0e0 e lo poiniamo in d0
        move.w        6(a1),d0

        ADDQ.L        #4,RIMTABPOINT        ; Fai puntare alla longword successiva
        MOVE.L        RIMTABPOINT(PC),A0 ; indirizzo contenuto in long RIMTABPOINT
                                  ; copiato in a0
        CMP.L        #FINERIMBALZTAB-4,A0 ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTART2               ; non ancora? allora continua
        MOVE.L        #RIMBALZTAB-4,RIMTABPOINT ; Riparti a puntare dalla prima long
NOBSTART2:
        MOVE.l        (A0),d1               ; copia la long dalla tabella in d1

        sub.l        d1,d0                ; sottraiamo il valore attualmente preso dalla
                                  ; tabella, facendo scorrere la figura SU o GIU.

        LEA         BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        MOVEQ         #1,D1                ; numero di bitplanes -1 (qua sono 3)
POINTBP2:
        move.w        d0,6(a1)          ; copia la word BASSA dell'indirizzo del plane
        swap         d0                 ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)          ; copia la word ALTA dell'indirizzo del plane
        swap         d0                 ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #80*256,d0         ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
;        dbra        d1,POINTBP2         ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


RIMTABPOINT:                         ; Questa longword "PUNTA" a RIMBALZTAB, ossia
        dc.l         RIMBALZTAB-4         ; contiene l'indirizzo di RIMBALZTAB. Terra'
                                  ; l'indirizzo del'ultima long "letta" dentro
                                  ; la tabella. (qua inizia da RIMORTAB-4 in
                                  ; quanto Lampeggio inizia con un ADDQ.L #4,C..
                                  ; serve per "bilanciare" la prima istruzione.

;        La tabella con i valori "precalcolati" del rimbalzo

RIMBALZTAB:
        dc.l         0,0,80,80,80,80,80,80,80,80,80                          ; in cima
        dc.l         80,80,2*80,2*80
        dc.l         2*80,2*80,2*80,2*80,2*80                          ; acceleriamo
        dc.l         3*80,3*80,3*80,3*80,3*80
        dc.l         3*80,3*80,3*80,3*80,3*80
        dc.l         2*80,2*80,2*80,2*80,2*80                          ; deceleriamo
        dc.l         2*80,2*80,80,80
        dc.l         80,80,80,80,80,80,80,80,80,0,0,0,0,0,0,0         ; in fondo
        dc.l         -80,-80,-80,-80,-80,-80,-80,-80,-80
        dc.l         -80,-80,-2*80,-2*80
        dc.l         -2*80,-2*80,-2*80,-2*80,-2*80
        dc.l         -3*80,-3*80,-3*80,-3*80,-3*80                          ; acceleriamo
        dc.l         -3*80,-3*80,-3*80,-3*80,-3*80
        dc.l         -2*80,-2*80,-2*80,-2*80,-2*80                          ; deceleriamo
        dc.l         -2*80,-2*80,-80,-80
        dc.l         -80,-80,-80,-80,-80,-80,-80,-80,-80,0,0,0,0,0         ; in cima
FINERIMBALZTAB:
```

```
;*********************************************************************
;*  fa una scacchiera con quadrati di 16 pixel di lato (Lezione6q.s)       *
;*********************************************************************

GRIGLIA3:
        LEA        BITPLANEGRIGLIA,a0        ; Indirizzo bitplane destinazione

        MOVEQ       #8-1,d0                ; 8 coppie di quadretti alti 16 pixel
                                  ; 8*2*16=256 riempimento completo dello schermo
FaiCoppia3:
        move.l       #(20*16)-1,d1        ; 20 longwords per riempire 1 linea (640 pixel)
                                  ; 16 linee da riempire
FaiUNO3:
        move.l       #%11111111111111110000000000000000,(a0)+
                                       ; lunghezza quadretto ad 1 = 16 pixel
                                       ; quadretto azzerato = 16 pixel
        dbra        d1,FaiUNO3                ; fai 16 linee #.#.#.#.#.#.#.#.#.#

        move.l       #(20*16)-1,d1        ; 20 lingwords per riempire 1 linea (640 pixel)
                                  ; 16 linee da riempire
FaiALTRO3:
        move.l       #%00000000000000001111111111111111,(a0)+
                                       ; lunghezza quadretto azzerato = 16
                                       ; quadretto ad 1 = 16 pixel
        dbra        d1,FaiAltro3               ; fai 8 linee .#.#.#.#.#.#.#.#.#.

        DBRA        d0,FaiCoppia3                ; fai 8 coppie di quadretti
                                     ; #.#.#.#.#.#.#.#.#.#
        rts                                  ; .#.#.#.#.#.#.#.#.#.

; *********************************************************************
; *                ROUTINE CHE SUONA MUSICHE SOUNDTRACKER/PROTRACKER          *
; *********************************************************************

        include        "music.s"        ; routine 100% funzionante su tutti gli Amiga


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8e,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$30                ; DdfStart (modificato per SCROLL)
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102                ; BplCon1
        dc.b        0                ; byte "alto" inutilizzato del $dff102
MIOBPCON1:
        dc.b        0                ; byte "basso" utilizzato del $dff102
        dc.w        $104,0                ; BplCon2
        dc.w        $108,40-2        ; Bpl1Mod (40 per la figura larga 640, il -2
        dc.w        $10a,40-2        ; Bpl2Mod (per bilanciare il DDFSTART

                    ; 5432109876543210
        dc.w        $100,%0010001000000000        ; bit 12 - 1 bitplane LOWRES

BPLPOINTERS:
```

```
        dc.w $e0,0,$e2,0        ; primo bitplane
BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ; secondo bitplane

        dc.w        $180,$113        ; color0 - QUADRATO SCURO
        dc.w        $182,$bb5        ; color1 - SCRITTE+quadrato scuro
        dc.w        $184,$225        ; color2 - QUADRATO CHIARO
        dc.w        $186,$bb5        ; color3 - SCRITTE+quadrato chiaro

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


;        Il FONT caratteri 8x8

FONT:
        incbin        "metal.fnt"
;        incbin        "normal.fnt"
;        incbin        "nice.fnt"

; ***************************************************************************
; *                                MUSICA PROTRACKER                        *
; ***************************************************************************

mt_data:
        incbin        "mod.purple-shades"


        SECTION        MIOPLANE,BSS_C

BITPLANEGRIGLIA:
        ds.b        80*256        ; un bitplane 640x256


        ds.b        80*100
BITPLANETESTO:
        ds.b        80*256        ; un bitplane 640x256


        end
```

Alle volte mettere insieme routines di poco effetto genera un bel risultato.

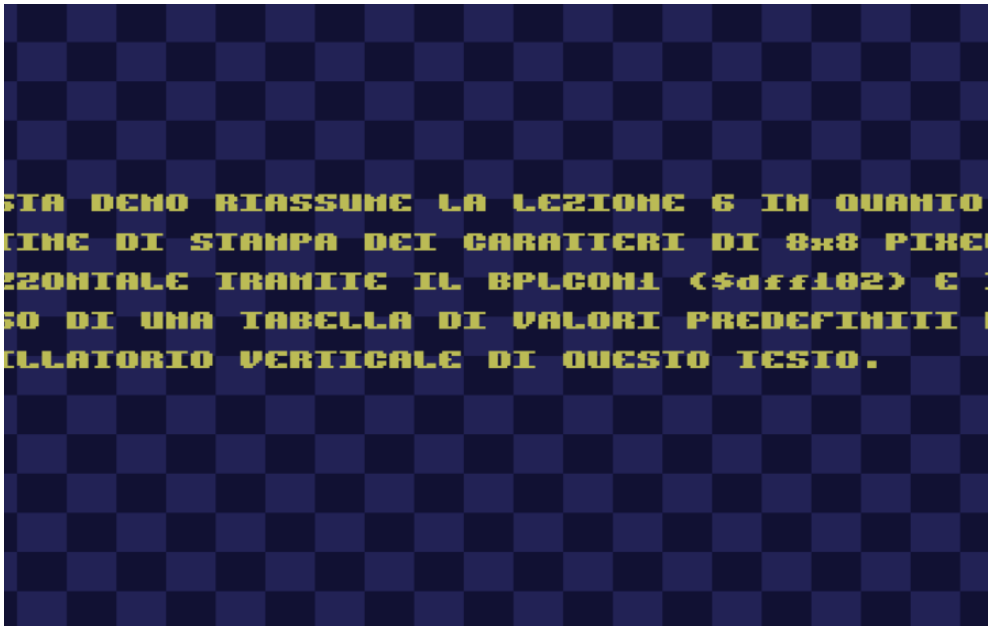


Figura 22.9: Lezione 6r

CAPITOLO 23

LEZIONE 7

23.1 Lezione7a

```
; Lezione7a.s                VISUALIZZAZIONE DI UNO SPRITE


        SECTION        CiriCop,CODE

Inizio:
        move.l          4.w,a6                  ; Execbase
        jsr            -$78(a6)          ; Disable
        lea            GfxName(PC),a1          ; Nome lib
        jsr            -$198(a6)          ; OpenLibrary
        move.l          d0,GfxBase
        move.l          d0,a6
        move.l          $26(a6),OldCop          ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L          #BITPLANE,d0           ; dove puntare
        LEA            BPLPOINTERS,A1          ; puntatori COP
        move.w          d0,6(a1)
        swap           d0
        move.w          d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L          #MIOSPRITE,d0                  ; indirizzo dello sprite in d0
        LEA            SpritePointers,a1          ; Puntatori in copperlist
        move.w          d0,6(a1)
        swap           d0
        move.w          d0,2(a1)

        move.l          #COPPERLIST,$dff080          ; nostra COP
        move.w          d0,$dff088                  ; START COP
        move.w          #0,$dff1fc                  ; NO AGA!
```

```
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist
```

```
; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $30        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $3d        ; $30+13=$3d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w       %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w       %0000000000000000,%0000011001100000
 dc.w       %0000000000000000,%0000001001000000
 dc.w       %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w       %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w       %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w       %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w       %0000011111100000,%0000011111100000
 dc.w       %0000011111100000,%0001111001111000
 dc.w       %0000001111000000,%0011101111011100
 dc.w       %0000000110000000,%0011000110001100
 dc.w       %0000000000000000,%1111000000001111
 dc.w       %0000000000000000,%1111000000001111
 dc.w       0,0         ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

Questo e' il primo sprite che controlliamo nel corso, potete facilmente definirne uno vostro cambiando i suoi 2 piani , che in questo listato sono definiti in binario; il colore risultante dalle varie sovrapposizioni binarie puo' essere intuito leggendo il commento a fianco dello sprite. I colori dello sprite 0 sono definiti dai registri del COLOR 17,18 e 19:

```
        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO
```

Per cambiare la posizione dello sprite, agite sui suoi primi byte:

```
MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $30        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $3d        ; $30+13=$3d        ; posizione verticale di fine sprite
        dc.b $00
```

Basta ricordarsi queste due cose:

1) L'angolo in alto a sinistra dello schermo non e' la posizione $00,$00 infatti lo schermo con l'overscan puo' essere piu' largo; nel caso dello schermo di larghezza normale la posizione orizzontale iniziale (HSTART) puo'

```
andare da $40 a $d8, altrimenti lo sprite viene "tagliato" o va proprio fuori
dallo schermo visibile. Allo stesso modo la posizione verticale iniziale, ossia
il VSTART, va selezionato a partire da $2c, cioe' dall'inizio della finestra
video definita in DIWSTART (che qua e' $2c81).
Per posizionare nello schermo 320x256 lo sprite, per esempio alla cordinata
centrale 160,128 bisogna tener conto che la prima coordinata in alto a sinistra
e' $40,$2c anziche' 0,0 per cui bisogna sommare $40 alla coordinata X e $2c
alla coordinata Y.
Infatti $40+160, $2c+128, corrispondono alla coordinata 160,128 di uno schermo
320x256 non overscan.
Non avendo ancora il controllo della posizione orizzontale a livello di 1
pixel, ma ogni 2 pixel, dobbiamo sommare non 160, ma 160/2 all'inizio per
individuare il centro dello schermo:

HSTART:
        dc.b $40+(160/2)        ; posizionato al centro dello schermo

Cosi' per altre coordinate orizzontali, ad esempio la posizione 50:

        dc.b $40+(50/2)

Piu' avanti vedremo come posizionare orizzontalmente 1 pixel alla volta.

2) La posizione orizzontale si puo' variare da sola per spostare a destra e a
sinistra uno sprite, mentre se si intende spostare lo sprite in alto o in basso
e' necessario ogni volta agire su due byte, ossia su VSTART e VSTOP, cioe' la
posizione verticale di inizio e di fine sprite. Infatti, mentre la larghezza di
uno sprite e' sempre 16, per cui determinata la posizione orizzontale di inizio
la posizione di fine e' sempre 16 pixel piu' a destra, per quanto riguarda la
lunghezza in verticale, essendo a piacere, e' necessario definirla comunicando
la posizione di inizio e di fine ogni volta, per cui se vogliamo spostare lo
sprite in alto dobbiamo sottrarre 1 sia a VSTART che a VSTOP, se vogliamo
spostarlo in basso e' necessario invece aggiungere 1 ad entrambi.
Se per esempio si vuole modificare il VSTART in $55, per determinare VSTOP
occorrera' sommare la lunghezza dello sprite (questo e' alto 13 linee) a
VSTART, dunque $55+13=$62.

Spostate lo sprite in varie posizioni dello schermo per verificare se avete
capito o se avete solo l'illusione di aver capito.
Non dimenticatevi che HSTART fa spostare di 2 pixel ogni volta e non di 1
pixel come potrebbe sembrare.
```

23.2 Lezione7b

```
; Lezione7b.s        VISUALIZZAZIONE DI UNO SPRITE - TASTO DESTRO PER MUOVERLO


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"
```

Figura 23.1: Lezione 7a

```
        MOVE.L      #BITPLANE,d0        ; dove puntare
        LEA       BPLPOINTERS,A1        ; puntatori COP
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L      #MIOSPRITE,d0              ; indirizzo dello sprite in d0
        LEA       SpritePointers,a1        ; Puntatori in copperlist
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)

        move.l      #COPPERLIST,$dff080        ; nostra COP
        move.w      d0,$dff088                ; START COP
        move.w      #0,$dff1fc                ; NO AGA!
        move.w      #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b      #$ff,$dff006        ; Linea 255?
        bne.s       mouse

        btst       #2,$dff016        ; Tasto destro del mouse premuto?
        bne.s       Aspetta               ; se no, salta la routine che muove lo sprite

        bsr.s       MuoviSprite        ; Muovi lo sprite 0 a destra

Aspetta:
        cmpi.b      #$ff,$dff006        ; linea 255?
        beq.s       Aspetta
```

```
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine sposta a destra lo sprite agendo sul suo byte HSTART, ossia
; il byte della sua posizione X. Da notare che scorre di 2 pixel ogni volta


MuoviSprite:
        addq.b        #1,HSTART        ; (come scrivere addq.b #1,MIOSPRITE+1)
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
```

```
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $30        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $3d        ; $30+13=$3d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end

Si puo' facilmente muovere lo sprite, provate con queste modifiche per la
routine MuoviSprite:


        subq.b        #1,HSTART        ; Spostamento a sinistra dello sprite

*

        ADDQ.B        #1,VSTART        ; \ sposta lo sprite in basso
        ADDQ.B        #1,VSTOP        ; / (bisogna agire sia su VSTART che su VSTOP!)

*
        SUBQ.B        #1,VSTART        ; \ sposta lo sprite in alto
        SUBQ.B        #1,VSTOP        ; / (bisogna agire sia su VSTART che su VSTOP!)

*

        ADDQ.B        #1,HSTART        ;\
        ADDQ.B        #1,VSTART        ; \ sposta in diagonale basso-destra
        ADDQ.B        #1,VSTOP        ; /
```

```
*
        SUBQ.B        #1,HSTART        ;\
        ADDQ.B        #1,VSTART        ; \ sposta  in diagonale basso-sinistra
        ADDQ.B        #1,VSTOP        ; /

*
        ADDQ.B        #1,HSTART        ;\
        SUBQ.B        #1,VSTART        ; \ sposta in diagonale alto-destra
        SUBQ.B        #1,VSTOP        ; /

*
        SUBQ.B        #1,HSTART        ;\
        SUBQ.B        #1,VSTART        ; \ sposta  in diagonale alto-sinistra
        SUBQ.B        #1,VSTOP        ; /

*
Provate poi a cambiare il valore aggiunto/sottratto per fare traiettorie piu'
insolite.
        SUBQ.B        #3,HSTART        ;\
        SUBQ.B        #1,VSTART        ; \ sposta  in diagonale alto-molto sinistra
        SUBQ.B        #1,VSTOP        ; /

Eccetera Eccetera.
```

23.3 Lezione7c

```
; Lezione7c.s        UNO SPRITE MOSSO ORIZZONTALMENTE USANDO UNA TABELLA DI VALORI
;                (ossia di coordinate orizzontali) PRESTABILITI.


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
```

```
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviSprite        ; Muovi lo sprite 0 orizzontalmente

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine sposta  lo sprite agendo sul suo byte HSTART, ossia
; il byte della sua posizione X, immettendoci delle coordinate gia' stabilite
; nella tabella TABX. Agendo solo su HSTART, lo scorrimento e' a scatti di 2
; pixel ogni volta e non di 1 pixel alla volta, per cui e' leggermente
; "scattoso" specialmente nei rallentamenti. Renderemo piu' fluido lo scroll
; orizzontale in seguito, rendendolo fedele al singolo pixel.

MuoviSprite:
        ADDQ.L        #1,TABXPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTART        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT ; Riparti a puntare dalla prima long
NOBSTART:
        MOVE.b        (A0),HSTART        ; copia il byte dalla tabella ad HSTART
        rts

TABXPOINT:
        dc.l        TABX-1                ; NOTA: i valori della tabella qua sono bytes,
                                ; dunque lavoriamo con un ADDQ.L #1,TABXPOINT
                                ; e non #2 come per quando sono word o con #4
```

```
                                ; come quando sono longword.

; Tabella con coordinate X dello sprite precalcolate.
; Da notare che la posizione X per far entrare lo sprite nella finestra video
; deve essere compresa tra $40 e $d8, infatti nella tabella ci sono byte non
; piu' grandi di $d8 e non piu' piccoli di $40.

TABX:
        dc.b        $41,$43,$46,$48,$4A,$4C,$4F,$51,$53,$55,$58,$5A ; 200 valori
        dc.b        $5C,$5E,$61,$63,$65,$67,$69,$6B,$6E,$70,$72,$74
        dc.b        $76,$78,$7A,$7C,$7E,$80,$82,$84,$86,$88,$8A,$8C
        dc.b        $8E,$90,$92,$94,$96,$97,$99,$9B,$9D,$9E,$A0,$A2
        dc.b        $A3,$A5,$A7,$A8,$AA,$AB,$AD,$AE,$B0,$B1,$B2,$B4
        dc.b        $B5,$B6,$B8,$B9,$BA,$BB,$BD,$BE,$BF,$C0,$C1,$C2
        dc.b        $C3,$C4,$C5,$C5,$C6,$C7,$C8,$C9,$C9,$CA,$CB,$CB
        dc.b        $CC,$CC,$CD,$CD,$CE,$CE,$CE,$CF,$CF,$CF,$CF,$D0
        dc.b        $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0,$CF,$CF,$CF
        dc.b        $CF,$CE,$CE,$CE,$CD,$CD,$CC,$CC,$CB,$CB,$CA,$C9
        dc.b        $C9,$C8,$C7,$C6,$C5,$C5,$C4,$C3,$C2,$C1,$C0,$BF
        dc.b        $BE,$BD,$BB,$BA,$B9,$B8,$B6,$B5,$B4,$B2,$B1,$B0
        dc.b        $AE,$AD,$AB,$AA,$A8,$A7,$A5,$A3,$A2,$A0,$9E,$9D
        dc.b        $9B,$99,$97,$96,$94,$92,$90,$8E,$8C,$8A,$88,$86
        dc.b        $84,$82,$80,$7E,$7C,$7A,$78,$76,$74,$72,$70,$6E
        dc.b        $6B,$69,$67,$65,$63,$61,$5E,$5C,$5A,$58,$55,$53
        dc.b        $51,$4F,$4C,$4A,$48,$46,$43,$41
FINETABX:



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO
```

```
        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end

I movimenti complessi e realistici si fanno con le tabelle!
Provate a sostituire la tabella corrente con questa, ed invertirete il rimbalzo
dello sprite. (Amiga+b+c+i per copiare), (amiga+b+x per cancellare un pezzo)

TABX:
        dc.b        $CF,$CD,$CA,$C8,$C6,$C4,$C1,$BF,$BD,$BB,$B8,$B6 ; 200 valori
        dc.b        $B4,$B2,$AF,$AD,$AB,$A9,$A7,$A5,$A2,$A0,$9E,$9C
        dc.b        $9A,$98,$96,$94,$92,$90,$8E,$8C,$8A,$88,$86,$84
        dc.b        $82,$80,$7E,$7C,$7A,$79,$77,$75,$73,$72,$70,$6E
        dc.b        $6D,$6B,$69,$68,$66,$65,$63,$62,$60,$5F,$5E,$5C
        dc.b        $5B,$5A,$58,$57,$56,$55,$53,$52,$51,$50,$4F,$4E
        dc.b        $4D,$4C,$4B,$4B,$4A,$49,$48,$47,$47,$46,$45,$45
        dc.b        $44,$44,$43,$43,$42,$42,$42,$41,$41,$41,$41,$40
        dc.b        $40,$40,$40,$40,$40,$40,$40,$40,$40,$41,$41,$41
        dc.b        $41,$42,$42,$42,$43,$43,$44,$44,$45,$45,$46,$47
        dc.b        $47,$48,$49,$4A,$4B,$4B,$4C,$4D,$4E,$4F,$50,$51
        dc.b        $52,$53,$55,$56,$57,$58,$5A,$5B,$5C,$5E,$5F,$60
        dc.b        $62,$63,$65,$66,$68,$69,$6B,$6D,$6E,$70,$72,$73
        dc.b        $75,$77,$79,$7A,$7C,$7E,$80,$82,$84,$86,$88,$8A
        dc.b        $8C,$8E,$90,$92,$94,$96,$98,$9A,$9C,$9E,$A0,$A2
        dc.b        $A5,$A7,$A9,$AB,$AD,$AF,$B2,$B4,$B6,$B8,$BB,$BD
        dc.b        $BF,$C1,$C4,$C6,$C8,$CA,$CD,$CF
FINETABX:
```

Ora sostituitela con questa, che fa oscillare da entrambe le parti lo sprite.

```
TABX:
        dc.b        $91,$93,$96,$98,$9A,$9C,$9F,$A1,$A3,$A5,$A7,$A9 ; 200 valori
        dc.b        $AC,$AE,$B0,$B2,$B4,$B6,$B8,$B9,$BB,$BD,$BF,$C0
        dc.b        $C2,$C4,$C5,$C7,$C8,$CA,$CB,$CC,$CD,$CF,$D0,$D1
        dc.b        $D2,$D3,$D3,$D4,$D5,$D5,$D6,$D7,$D7,$D7,$D8,$D8
        dc.b        $D8,$D8,$D8,$D8,$D8,$D8,$D7,$D7,$D7,$D6,$D5,$D5
        dc.b        $D4,$D3,$D3,$D2,$D1,$D0,$CF,$CD,$CC,$CB,$CA,$C8
        dc.b        $C7,$C5,$C4,$C2,$C0,$BF,$BD,$BB,$B9,$B8,$B6,$B4
        dc.b        $B2,$B0,$AE,$AC,$A9,$A7,$A5,$A3,$A1,$9F,$9C,$9A
        dc.b        $98,$96,$93,$91,$8F,$8D,$8A,$88,$86,$84,$81,$7F
        dc.b        $7D,$7B,$79,$77,$74,$72,$70,$6E,$6C,$6A,$68,$67
        dc.b        $65,$63,$61,$60,$5E,$5C,$5B,$59,$58,$56,$55,$54
        dc.b        $53,$51,$50,$4F,$4E,$4D,$4D,$4C,$4B,$4B,$4A,$49
        dc.b        $49,$49,$48,$48,$48,$48,$48,$48,$48,$48,$49,$49
        dc.b        $49,$4A,$4B,$4B,$4C,$4D,$4D,$4E,$4F,$50,$51,$53
        dc.b        $54,$55,$56,$58,$59,$5B,$5C,$5E,$60,$61,$63,$65
        dc.b        $67,$68,$6A,$6C,$6E,$70,$72,$74,$77,$79,$7B,$7D
        dc.b        $7F,$81,$84,$86,$88,$8A,$8D,$8F
FINETABX:
```

23.4 Lezione7d

```
; Lezione7d.s        UNO SPRITE MOSSO VERTICALMENTE USANDO UNA TABELLA DI VALORI
;                (ossia di coordinate verticali) PRESTABILITI.
;                - Da notare che possiamo arrivare in basso fino alla linea $FF
;                  e non oltre.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr          -$78(a6)         ; Disable
        lea          GfxName(PC),a1         ; Nome lib
        jsr          -$198(a6)         ; OpenLibrary
        move.l          d0,GfxBase
        move.l          d0,a6
        move.l          $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L          #BITPLANE,d0         ; dove puntare
        LEA          BPLPOINTERS,A1        ; puntatori COP
        move.w          d0,6(a1)
        swap         d0
        move.w          d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L          #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA          SpritePointers,a1        ; Puntatori in copperlist
        move.w          d0,6(a1)
        swap         d0
        move.w          d0,2(a1)

        move.l          #COPPERLIST,$dff080         ; nostra COP
        move.w          d0,$dff088                ; START COP
        move.w          #0,$dff1fc                ; NO AGA!
```

```
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s         mouse

        bsr.s         MuoviSprite         ; Muovi lo sprite 0 verticalmente

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001         ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)         ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)         ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0

OldCop:
        dc.l         0

; Questa routine sposta in alto e in basso lo sprite agendo sui suoi byte
; VSTART e VSTOP, ossia i byte della sua posizione Y di inizio e fine,
; immettendoci delle coordinate gia' stabilite nella tabella TABY

MuoviSprite:
        ADDQ.L        #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTART        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dalla prima long
NOBSTART:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte dalla tabella in d0
        MOVE.b        d0,VSTART        ; copia il byte in VSTART
        ADD.B        #13,D0                ; Aggiungi la lunghezza dello sprite per
                                  ; determinare la posizione finale (VSTOP)
        move.b        d0,VSTOP        ; Muovi il valore giusto in VSTOP
        rts

TABYPOINT:
        dc.l         TABY-1                ; NOTA: i valori della tabella qua sono bytes,
                                  ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                                  ; e non #2 come per quando sono word o con #4
                                  ; come quando sono longword.

; Tabella con coordinate Y dello sprite precalcolate.
```

```
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $2c e $f2, infatti nella tabella ci sono byte non
; piu' grandi di $f2 e non piu' piccoli di $2c.

TABY:
        dc.b        $EE,$EB,$E8,$E5,$E2,$DF,$DC,$D9,$D6,$D3,$D0,$CD ; salto in
        dc.b        $CA,$C7,$C4,$C1,$BE,$BB,$B8,$B5,$B2,$AF,$AC,$A9 ; alto da
        dc.b        $A6,$A4,$A1,$9E,$9B,$98,$96,$93,$90,$8E,$8B,$88 ; record!
        dc.b        $86,$83,$81,$7E,$7C,$79,$77,$74,$72,$70,$6D,$6B ; 200 valori
        dc.b        $69,$66,$64,$62,$60,$5E,$5C,$5A,$58,$56,$54,$52
        dc.b        $51,$4F,$4D,$4B,$4A,$48,$47,$45,$44,$42,$41,$3F
        dc.b        $3E,$3D,$3C,$3A,$39,$38,$37,$36,$35,$34,$33,$33
        dc.b        $32,$31,$30,$30,$2F,$2F,$2E,$2E,$2D,$2D,$2D,$2C
        dc.b        $2C,$2C,$2C,$2C,$2C,$2C,$2C,$2C,$2C,$2D,$2D,$2D
        dc.b        $2E,$2E,$2F,$2F,$30,$30,$31,$32,$33,$33,$34,$35
        dc.b        $36,$37,$38,$39,$3A,$3C,$3D,$3E,$3F,$41,$42,$44
        dc.b        $45,$47,$48,$4A,$4B,$4D,$4F,$51,$52,$54,$56,$58
        dc.b        $5A,$5C,$5E,$60,$62,$64,$66,$69,$6B,$6D,$70,$72
        dc.b        $74,$77,$79,$7C,$7E,$81,$83,$86,$88,$8B,$8E,$90
        dc.b        $93,$96,$98,$9B,$9E,$A1,$A4,$A6,$A9,$AC,$AF,$B2
        dc.b        $B5,$B8,$BB,$BE,$C1,$C4,$C7,$CA,$CD,$D0,$D3,$D6
        dc.b        $D9,$DC,$DF,$E2,$E5,$E8,$EB,$EE
FINETABY:


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************
```

```
MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0         ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end

I movimenti complessi e realistici si fanno con le tabelle!
Provate a sostituire la tabella corrente con questa, ed otterrete un ondeggio
dello sprite. (Amiga+b+c+i per copiare), (amiga+b+x per cancellare un pezzo)


TABY:
        dc.b        $8E,$91,$94,$97,$9A,$9D,$A0,$A3,$A6,$A9,$AC,$AF ; ondeggio
        dc.b        $B2,$B4,$B7,$BA,$BD,$BF,$C2,$C5,$C7,$CA,$CC,$CE ; 200 valori
        dc.b        $D1,$D3,$D5,$D7,$D9,$DB,$DD,$DF,$E0,$E2,$E3,$E5
        dc.b        $E6,$E7,$E9,$EA,$EB,$EC,$EC,$ED,$EE,$EE,$EF,$EF
        dc.b        $EF,$EF,$F0,$EF,$EF,$EF,$EF,$EE,$EE,$ED,$EC,$EC
        dc.b        $EB,$EA,$E9,$E7,$E6,$E5,$E3,$E2,$E0,$DF,$DD,$DB
        dc.b        $D9,$D7,$D5,$D3,$D1,$CE,$CC,$CA,$C7,$C5,$C2,$BF
        dc.b        $BD,$BA,$B7,$B4,$B2,$AF,$AC,$A9,$A6,$A3,$A0,$9D
        dc.b        $9A,$97,$94,$91,$8E,$8B,$88,$85,$82,$7F,$7C,$79
        dc.b        $76,$73,$70,$6D,$6A,$68,$65,$62,$5F,$5D,$5A,$57
        dc.b        $55,$52,$50,$4E,$4B,$49,$47,$45,$43,$41,$3F,$3D
        dc.b        $3C,$3A,$39,$37,$36,$35,$33,$32,$31,$30,$30,$2F
        dc.b        $2E,$2E,$2D,$2D,$2D,$2D,$2C,$2D,$2D,$2D,$2D,$2E
        dc.b        $2E,$2F,$30,$30,$31,$32,$33,$35,$36,$37,$39,$3A
        dc.b        $3C,$3D,$3F,$41,$43,$45,$47,$49,$4B,$4E,$50,$52
        dc.b        $55,$57,$5A,$5D,$5F,$62,$65,$68,$6A,$6D,$70,$73
        dc.b        $76,$79,$7C,$7F,$82,$85,$88,$8B
FINETABY:
```

23.5 Lezione7e

```
; Lezione7e.s        UNO SPRITE MOSSO SIA VERTICALMENTE CHE ORIZZONTALMENTE
;                  USANDO DUE TABELLE DI VALORI (ossia di coordinate verticali
;                 e orizzontali) PRESTABILITI.
;                 Nella nota finale viene spiegato come farsi proprie tabelle.


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1         ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0         ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviSpriteX        ; Muovi lo sprite 0 orizzontalmente
        bsr.w        MuoviSpriteY        ; Muovi lo sprite 0 verticalmente

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
```

```
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; In questo esempio sono state incluse le routine e le tabelle dei due esempi
; precedenti, dunque agiamo sia sulla x che sulla y dello sprite.
; Essendo le due tabelle X ed Y entrambe formate da 200 coordinate, si
; verifica sempre la stessa "accoppiata" di coordinate:
; valore 1 della tabella X + valore 1 della tabella Y
; valore 2 della tabella X + valore 2 della tabella Y
; valore 3 della tabella X + valore 3 della tabella Y
; ....
; Dunque il risultato e' che lo sprite ondeggia in diagonale, come abbiamo gia'
; visto mettendo insieme addq.b #1,HSTART e addq.b #1,VSTART/VSTOP.


; Questa routine sposta  lo sprite agendo sul suo byte HSTART, ossia
; il byte della sua posizione X, immettendoci delle coordinate gia' stabilite
; nella tabella TABX. (scatti di 2 pixel minimo e non 1 pixel)

MuoviSpriteX:
        ADDQ.L        #1,TABXPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT ; Riparti a puntare dal primo byte-1
NOBSTARTX:
        MOVE.b        (A0),HSTART        ; copia il byte dalla tabella ad HSTART
        rts

TABXPOINT:
        dc.l        TABX-1                ; NOTA: i valori della tabella qua sono bytes,
                                  ; dunque lavoriamo con un ADDQ.L #1,TABXPOINT
                                  ; e non #2 come per quando sono word o con #4
                                  ; come quando sono longword.

; Tabella con coordinate X dello sprite precalcolate.
; Da notare che la posizione X per far entrare lo sprite nella finestra video
; deve essere compresa tra $40 e $d8, infatti nella tabella ci sono byte non
; piu' grandi di $d8 e non piu' piccoli di $40.

TABX:
        dc.b        $91,$93,$96,$98,$9A,$9C,$9F,$A1,$A3,$A5,$A7,$A9 ; 200 valori
        dc.b        $AC,$AE,$B0,$B2,$B4,$B6,$B8,$B9,$BB,$BD,$BF,$C0
        dc.b        $C2,$C4,$C5,$C7,$C8,$CA,$CB,$CC,$CD,$CF,$D0,$D1
        dc.b        $D2,$D3,$D3,$D4,$D5,$D5,$D6,$D7,$D7,$D7,$D8,$D8
        dc.b        $D8,$D8,$D8,$D8,$D8,$D8,$D7,$D7,$D7,$D6,$D5,$D5
        dc.b        $D4,$D3,$D3,$D2,$D1,$D0,$CF,$CD,$CC,$CB,$CA,$C8
        dc.b        $C7,$C5,$C4,$C2,$C0,$BF,$BD,$BB,$B9,$B8,$B6,$B4
        dc.b        $B2,$B0,$AE,$AC,$A9,$A7,$A5,$A3,$A1,$9F,$9C,$9A
        dc.b        $98,$96,$93,$91,$8F,$8D,$8A,$88,$86,$84,$81,$7F
        dc.b        $7D,$7B,$79,$77,$74,$72,$70,$6E,$6C,$6A,$68,$67
```

```
        dc.b        $65,$63,$61,$60,$5E,$5C,$5B,$59,$58,$56,$55,$54
        dc.b        $53,$51,$50,$4F,$4E,$4D,$4D,$4C,$4B,$4B,$4A,$49
        dc.b        $49,$49,$48,$48,$48,$48,$48,$48,$48,$48,$49,$49
        dc.b        $49,$4A,$4B,$4B,$4C,$4D,$4D,$4E,$4F,$50,$51,$53
        dc.b        $54,$55,$56,$58,$59,$5B,$5C,$5E,$60,$61,$63,$65
        dc.b        $67,$68,$6A,$6C,$6E,$70,$72,$74,$77,$79,$7B,$7D
        dc.b        $7F,$81,$84,$86,$88,$8A,$8D,$8F
FINETABX:


        even        ; pareggia l'indirizzo seguente


; Questa routine sposta in alto e in basso lo sprite agendo sui suoi byte
; VSTART e VSTOP, ossia i byte della sua posizione Y di inizio e fine,
; immettendoci delle coordinate gia' stabilite nella tabella TABY

MuoviSpriteY:
        ADDQ.L       #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L       TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                  ; copiato in a0
        CMP.L       #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S       NOBSTARTY        ; non ancora? allora continua
        MOVE.L       #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte (-1)
NOBSTARTY:
        moveq       #0,d0                ; Pulisci d0
        MOVE.b       (A0),d0                ; copia il byte dalla tabella in d0
        MOVE.b       d0,VSTART        ; copia il byte in VSTART
        ADD.B       #13,D0                ; Aggiungi la lunghezza dello sprite per
                                  ; determinare la posizione finale (VSTOP)
        move.b       d0,VSTOP        ; Muovi il valore giusto in VSTOP
        rts

TABYPOINT:
        dc.l        TABY-1                ; NOTA: i valori della tabella qua sono bytes,
                                  ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                                  ; e non #2 come per quando sono word o con #4
                                  ; come quando sono longword.

; Tabella con coordinate Y dello sprite precalcolate.
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $2c e $f2, infatti nella tabella ci sono byte non
; piu' grandi di $f2 e non piu' piccoli di $2c.

TABY:
        dc.b        $8E,$91,$94,$97,$9A,$9D,$A0,$A3,$A6,$A9,$AC,$AF ; ondeggio
        dc.b        $B2,$B4,$B7,$BA,$BD,$BF,$C2,$C5,$C7,$CA,$CC,$CE ; 200 valori
        dc.b        $D1,$D3,$D5,$D7,$D9,$DB,$DD,$DF,$E0,$E2,$E3,$E5
        dc.b        $E6,$E7,$E9,$EA,$EB,$EC,$EC,$ED,$EE,$EE,$EF,$EF
        dc.b        $EF,$EF,$F0,$EF,$EF,$EF,$EF,$EE,$EE,$ED,$EC,$EC
        dc.b        $EB,$EA,$E9,$E7,$E6,$E5,$E3,$E2,$E0,$DF,$DD,$DB
        dc.b        $D9,$D7,$D5,$D3,$D1,$CE,$CC,$CA,$C7,$C5,$C2,$BF
        dc.b        $BD,$BA,$B7,$B4,$B2,$AF,$AC,$A9,$A6,$A3,$A0,$9D
        dc.b        $9A,$97,$94,$91,$8E,$8B,$88,$85,$82,$7F,$7C,$79
        dc.b        $76,$73,$70,$6D,$6A,$68,$65,$62,$5F,$5D,$5A,$57
        dc.b        $55,$52,$50,$4E,$4B,$49,$47,$45,$43,$41,$3F,$3D
        dc.b        $3C,$3A,$39,$37,$36,$35,$33,$32,$31,$30,$30,$2F
        dc.b        $2E,$2E,$2D,$2D,$2D,$2D,$2C,$2D,$2D,$2D,$2D,$2E
        dc.b        $2E,$2F,$30,$30,$31,$32,$33,$35,$36,$37,$39,$3A
        dc.b        $3C,$3D,$3F,$41,$43,$45,$47,$49,$4B,$4E,$50,$52
        dc.b        $55,$57,$5A,$5D,$5F,$62,$65,$68,$6A,$6D,$70,$73
        dc.b        $76,$79,$7C,$7F,$82,$85,$88,$8b
```

```
FINETABY:


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
```

```
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

Fino ad adesso abbiamo fatto andare lo sprite in orizzontale, in verticale, e in diagonale, ma mai gli abbiamo fatto fare curve. Ebbene basta modificare questo listato per fargli fare tutte le curve possibili, infatti possiamo variare le sue coordinate X ed Y tramite due tabelle. In questo listato sono riportate due tabelle di uguale lunghezza (200 valori) per cui ogni volta avvengono sempre le stesse "accoppiate" di coordinate X ed Y:

```
 valore 1 della tabella X + valore 1 della tabella Y
 valore 2 della tabella X + valore 2 della tabella Y
 valore 3 della tabella X + valore 3 della tabella Y
 ....
```

Dunque il risultato e' sempre la stessa oscillazione in diagonale.
Se pero' una delle due tabelle fosse piu' corta, questa ripartirebbe prima da capo dell'altra creando nuove oscillazioni, e ogni volta le due tabelle farebbero delle accoppiate XX ed YY diverse, ad esempio:

```
 valore 23 della tabella X + valore 56 della tabella Y
 valore 24 della tabella X + valore 57 della tabella Y
 valore 25 della tabella X + valore 58 della tabella Y
 ....
```

Queste accoppiate si tradurrebbero in oscillazioni curvilinee dello sprite

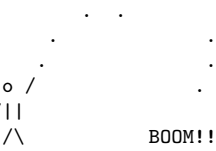
Provate a sostituire la tabella corrente delle coordinate XX con questa:
(Amiga+b+c+i per copiare), (amiga+b+x per cancellare un pezzo)

```
TABX:
        dc.b        $8A,$8D,$90,$93,$95,$98,$9B,$9E,$A1,$A4,$A7,$A9 ; 150 valori
        dc.b        $AC,$AF,$B1,$B4,$B6,$B8,$BA,$BC,$BF,$C0,$C2,$C4
        dc.b        $C6,$C7,$C8,$CA,$CB,$CC,$CD,$CE,$CE,$CF,$CF,$D0
        dc.b        $D0,$D0,$D0,$D0,$CF,$CF,$CE,$CE,$CD,$CC,$CB,$CA
        dc.b        $C8,$C7,$C6,$C4,$C2,$C0,$BF,$BC,$BA,$B8,$B6,$B4
        dc.b        $B1,$AF,$AC,$A9,$A7,$A4,$A1,$9E,$9B,$98,$95,$93
        dc.b        $90,$8D,$8A,$86,$83,$80,$7D,$7B,$78,$75,$72,$6F
        dc.b        $6C,$69,$67,$64,$61,$5F,$5C,$5A,$58,$56,$54,$51
        dc.b        $50,$4E,$4C,$4A,$49,$48,$46,$45,$44,$43,$42,$42
        dc.b        $41,$41,$40,$40,$40,$40,$40,$41,$41,$42,$42,$43
        dc.b        $44,$45,$46,$48,$49,$4A,$4C,$4E,$50,$51,$54,$56
        dc.b        $58,$5A,$5C,$5F,$61,$64,$67,$69,$6C,$6F,$72,$75
        dc.b        $78,$7B,$7D,$80,$83,$86
FINETABX:
```

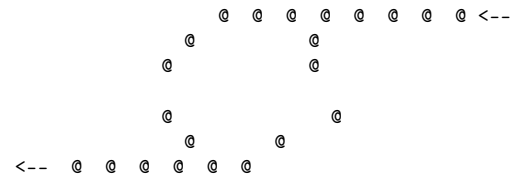
Ora potete ammirare lo sprite ondeggiare per lo schermo realisticamente e con un movimento variabile, a causa della differenza di lunghezza delle due tabelle

Con due tabelle, una per la posizione XX ed una per la posizione YY, vanno definiti i vari movimenti curvilinei dei giochi e delle dimostrazioni grafiche,

ad esempio il lancio di una bomba:



La curva percorsa dalla bomba lanciata dal protagonista del nostro gioco e'
stata simulata tramite il precalcolamento di essa in termini di XX ed YY.
Dato che il personaggio al momento del lancio poteva trovarsi in posizioni
diverse dello schermo, tutto spostato a destra o a sinistra, bastera'
aggiungere la posizione del protagonista lanciatore alle coordinate della
curva e la bomba partira' e cadra' nel posto giusto.
Oppure i movimenti di una squadriglia di astronavi nemiche:



Gli utilizzi delle coordinate nelle tabelle sono infiniti.

Vi starete chiedendo: ma le tabelle si fanno a mano calcolandosi ad occhio
l'onda?? Ebbene NO, esiste un comando dell'ASMONE, il "CS" (oppure "IS"), che
puo' bastare per fare le tabelle presenti in questo listato (infatti le ho
fatte proprio con questo comando!). Oppure se serve qualche tabella "speciale"
ci si puo' fare un programmino che la faccia.

Anticipiamo l'argomento "come farsi una tabella":
Il comando CS significa "CREATE SINUS", che per chi conosce la trigonometria
significa "TUTTO LI'?", mentre per chi non la conosce significa "COSA E'?".
Dato che questo deve essere solo un accenno, spieghero' soltanto come dare i
parametri al comando "CS" o "IS".

Il comando "CS" crea i valori in memoria dall'indirizzo o dalla label che viene
specificata, ad esempio se e' gia' presente una tab di 200 bytes alla label
TABX, se si crea all'indirizzo "TABX", dopo aver assemblato, un'altra tabella
di 200 bytes, questa sara' "sovrapposta" a quella precedente in memoria, ed
eseguendo il listato si vedra' l'effetto dell'ultima tabella creata.
Ma assemblando nuovamente viene riassemblata la tabella precedente, in quanto
non abbiamo cambiato il testo (dc.b $xx,$xx..).
Per salvare la tabella allora si puo' creare sopra un'altra di uguale grandezza
oppure si puo' fare un "buffer", ossia una zona di memoria dedicata alla
creazione e al salvataggio su disco della tabella.
Facciamo un esempio pratico: vogliamo fare una tabella particolare lunga 512
bytes, e la vogliamo salvare su disco per poterla ricaricare col comando
incbin cosi':

```
TABX:
        incbin        "TABELLA1"
```

Per fare la TABELLA1 da salvare dobbiamo prima crearci uno spazio vuoto di 512
byte dove crearla col comando "CS":

```
SPAZIO:
        dcb.b        512,0        ; 512 byte azzerati dove sara' creata la tabella
FINESPAZIO:
```

Una volta assemblato, creeremo la tabella definendo come destinazione "SPAZIO":

```
 DEST> SPAZIO
```

E naturalmente 512 valori da generare, di grandezza BYTE:

```
 AMOUNT> 512
 SIZE (B/W/L)> B
```

A questo punto avremo la tabella generata nei 512 bytes che vanno da SPAZIO:
a FINESPAZIO: , dunque dobbiamo salvare quel pezzo di memoria in un file.
Per questo esisite un comando dell'ASMONE, il "WB" (ossia Write Binary, cioe'
SCRIVI UN PEZZO DI MEMORIA). Per salvare la nostra tabella bastera' eseguire
queste operazioni:

1) Scrivere "WB" e definire il nome che si vuole dare al file, es "TABELLA1"
2) alla domanda BEG> (begin ossia da dove partire) scrivere SPAZIO
3) alla domanda END> (ossia FINE) scrivere FINESPAZIO

Otterremo in questo modo un file TABELLA1 lungo naturalmente 512 bytes che
conterra' la tabella, ricaricabile con l'INCBIN.

Il comando WB puo' essere applicato per salvare qualsiasi pezzo di memoria!
Potete provare a salvare uno sprite sprite e ricaricarlo con l'incbin.

L'altro sistema e' il comando "IS", ossia INSERT SINUS, inserisci il sinus
nel testo. In questo caso la tabella viene creata direttamente nel listato in
formato dc.b. Puo' essere comodo per piccole tabelle.
Basta posizionarsi col cursore dove si vuole che venga scritta la tabella, ad
esempio sotto la label "TABX:"; a questo punto di deve premere ESC per passare
alla linea di comandi e fare la tabella col comando "IS" anziche' "CS", la
procedura e i parametri da passare sono gli stessi.
Premendo nuovamente ESC troveremo la tabella fatta di dc.b sotto TABX:.

ma vediamo come CREARE una SONTAB usando il comando CS o IS dell'ASMONE:

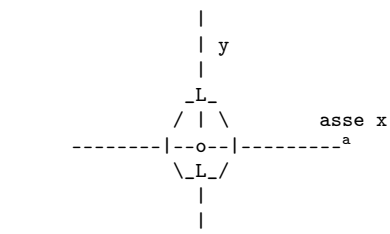
```
 DEST> indirizzo o label di destinazione, esempio: DEST>tabx
 BEG> angolo di inizio (0-360) (si possono dare anche valori superiori a 360)
 END> angolo di fine (0-360)
 AMOUNT> numero di valori da generare (esempio: 200 come in questo listato)
 AMPLITUDE> ampiezza, ossia valore piu' alto da raggiungere
 YOFFSET> offset (numero aggiunto a tutti i valori per spostare in "alto")
 SIZE (B/W/L)> dimensione dei valori (byte,word,long)
 MULTIPLIER> "moltiplicatore" (moltiplica l'ampiezza)
 HALF CORRECTION>Y/N                \ questi si occupano di "lisciare" l'onda
 ROUND CORRECTION>Y/N                / per "correggere" eventuali sbalzi.
```

Chi sa cosa sono il SENO ed il COSENO capira' al volo come fare, per chi non
lo sa posso dire che con BEG> ed END> si definisce l'angolo inizio e l'angolo
fine dell'onda, ossia la forma dell'onda, se questa comincera' calando e poi
risalendo, oppure se comincera' salendo e poi ricalando. Qua di seguito ci
sono degli esempi con il disegno della curva a fianco.

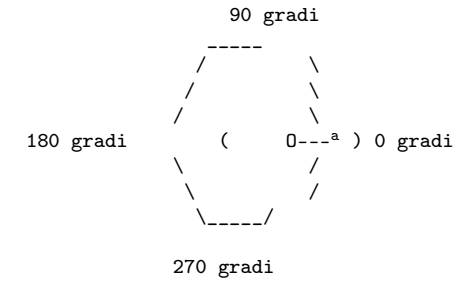
- Con AMOUNT> si decide quanti valori debba avere la tabella.
- Con AMPLITUDE si definisce l'ampiezza dell'onda, ossia il valore massimo che
  raggiungera' in alto, o in negativo, se e' presente la parte di curva

negativa.
- Con YOFFSET si decide di quanto "alzare" l'intera curva, ossia quanto si deve aggiungere ad ogni valore della tabella. Se per esempio una tabella fosse composta da 0,1,2,3,4,5,4,3,2,1,0 con un YOFFSET di 0, mettendo un YOFFSET di 10 otterremmo 10,11,12,13,14,15,14,13,12,11,10. Nel caso delle posizioni dello sprite, sappiamo che la X parte da $40 ed arriva a $d8, dunque l'YOFFSET sara' di $40, per traformare gli eventuali $00 in $40, gli $01 in $41 eccetera.
- Con "SIZE" definiamo se i valori della tabella saranno byte, word o longword. Nel caso delle coordinate dello sprite sono BYTE.
- Il MULTIPLIER> e' un moltiplicatore dell'ampiezza, se non si vuole moltiplicare basta definirlo come 1.

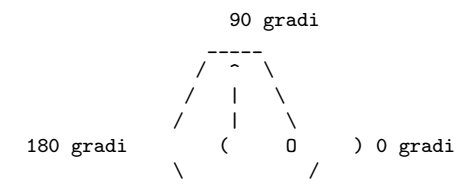
Ora rimane da chiarire come definire la "forma dell'onda", ossia la cosa piu' importante, e per questo possiamo usare solo BEG> ed END> che si riferiscono all'angolo inizio e all'angolo di fine di tale curva dal punto di vista trigonometrico. Per chi non conosce la trigonometria consiglio di studiarla un poco, anche perche' e' importante per le routines tridimensionali. Brevemente posso sintetizzare cosi': immaginatevi una circonferenza con un centro 0 e raggio come vi pare (per motivi tecnici il cerchio non e' tondo..) inserito negli assi cartesiani X ed Y, per cui il centro 0 si trova alla posizione 0,0: (ridisegnate su carta questi passaggi)



Supponiamo ora che sia per un momento un orologio ad una sola lancetta che vada all'indietro (che esempio contorto!) partendo da questa posizione:



(fate finta che sia un cerchio!!!) In pratica segna le 3. Al posto delle ore qua abbiamo i gradi formati dalla lancetta rispetto all'asse X, infatti quando segna le 12 e' a 90 gradi rispetto all'asse X:



```
                          \         /
                           \_____/

                         270 gradi
```

Allo stesso modo, questi sono 45 gradi:

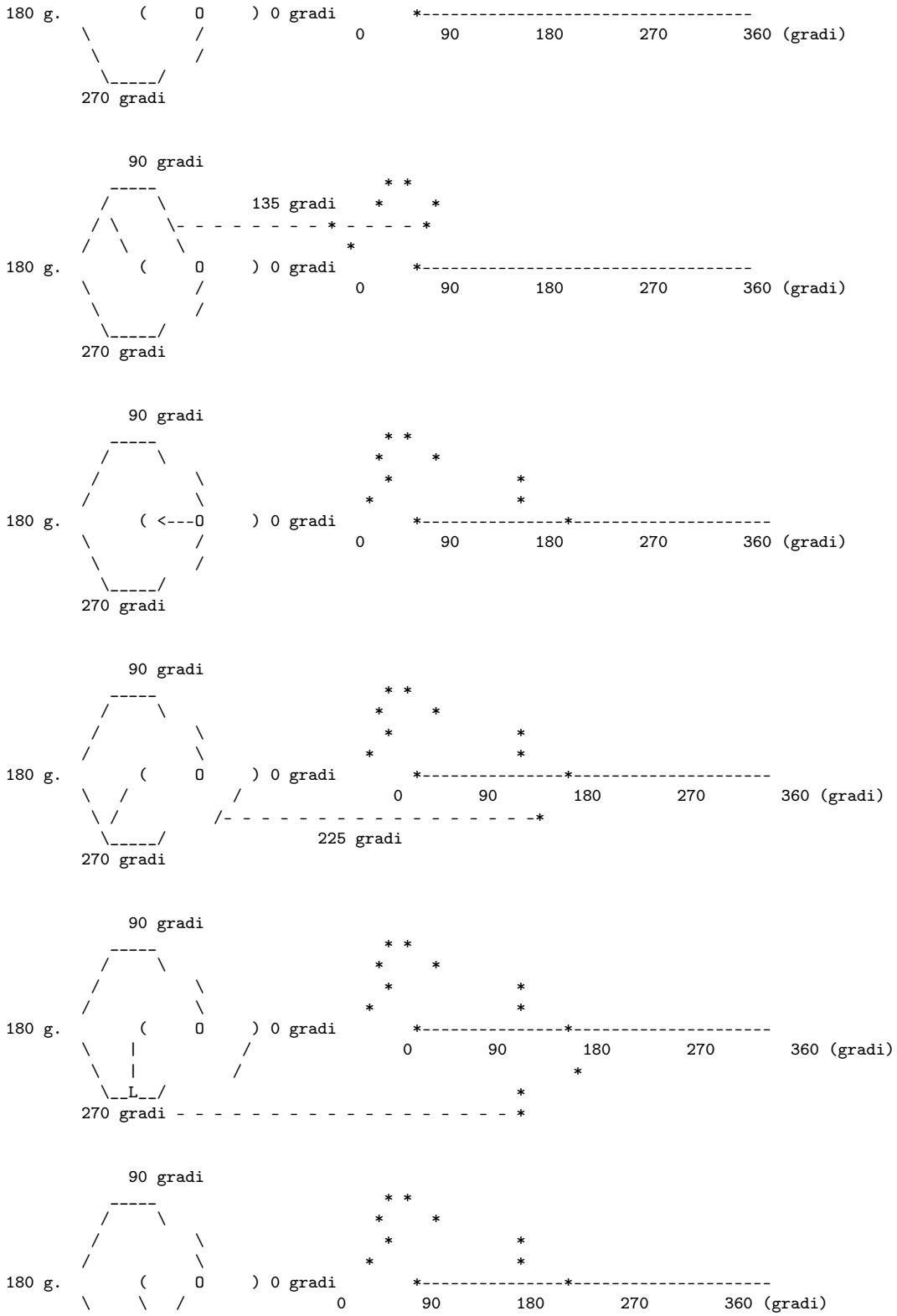
```
                              90 gradi
                             -----
                            /     \
                           /     / \
                          /     /   \
          180 gradi        (     O     ) 0 gradi (o anche 360, il giro completo)
                          \             /
                           \           /
                            \_____/

                         270 gradi
```

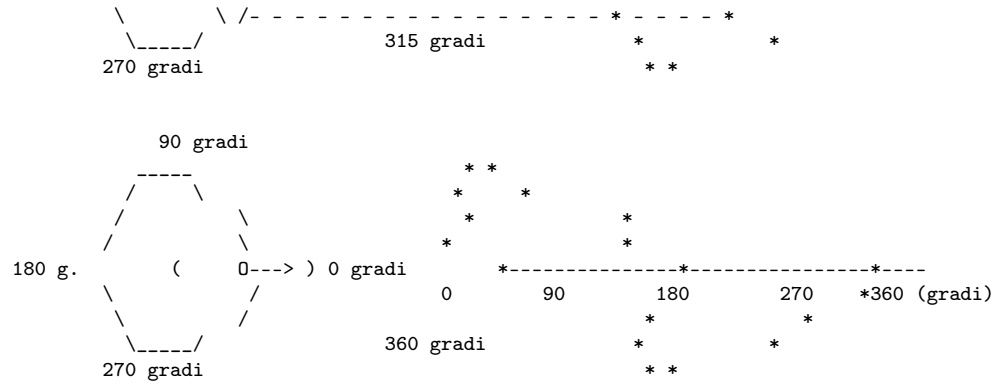
Ci siamo con questo balordo orologio che va al contrario e che ha i gradi al posto delle ore?? Ora veniamo al nesso con i BEG> ed END> del comando "CS". Disponendo di questo orologio, si puo' fare lo studio dell'andamento della funzione SENO (e COSENO, perche' no). Immaginiamo di far fare un giro completo alla lancetta, partendo da 0 gradi a 360, ossia la stessa posizione dopo un giro completo: se registriamo in un grafico accanto all'orologio i movimenti della punta della lancetta rispetto all'asse Y noteremo che parte da zero, poi sale fino alla massima altezza raggiunta ai 90 gradi, dopodiche' scende nuovamente ritornando a zero una volta giunto a 180 gradi, e continua a scendere sotto lo zero fino al minimo dei 270 gradi, per poi risalire fino allo zero iniziale dei 360 gradi (stessa posizione della partenza):

```
             90 gradi
            -----
           /         \
          /           \
         /             \
180 g.        (     O---ª ) 0 gradi        *-----------------------------------
         \             /                0        90        180        270        360 (gradi)
          \           /
           \_____/
         270 gradi
```

```
             90 gradi
            -----
           /         \         45 gradi
          /        / \- - - - - - - - *
         /     /   \                 *
180 g.        (     O     ) 0 gradi        *-----------------------------------
         \             /                0        90        180        270        360 (gradi)
          \           /
           \_____/
         270 gradi
```

```
             90 gradi
            ----- - - - - - - - - - - - *
           /  ^  \                     *
          /   |   \                   *
         /    |    \                 *
```


180 g.
0 gradi
0
90
180
270
360 (gradi)
270 gradi
90 gradi
135 gradi
180 g.
0 gradi
0
90
180
270
360 (gradi)
270 gradi
90 gradi
180 g.
0 gradi
0
90
180
270
360 (gradi)
270 gradi
90 gradi
180 g.
0 gradi
0
90
180
270
360 (gradi)
225 gradi
270 gradi
90 gradi
180 g.
0 gradi
0
90
180
270
360 (gradi)
270 gradi
90 gradi
180 g.
0 gradi
0
90
180
270
360 (gradi)



Spero di essere stato abbastanza chiaro per chi e' a digiuno di matematica:
per fare una curva che sale e scende basta dare come angolo inizio 0 e come
angolo di fine 180**!!!** Per fare una curva che scende e risale basta dare
come angolo inizio BEG> 180 e come angolo fine END> 360, cosi' per tutte le
altre curve. Cambiando AMPLITUDE, YOFFSET e MULTIPLIER farete curve piu'
lunghe e strette o piu' o meno lunghe. Si possono usare anche valori superiori
a 360 per utilizzare la curva del secondo "giro d'orologio", dato che la
funzione e' continua: /\/\/\/\/\/\/\/\/\/\.....

Facciamo degli esempi: (sotto il disegno viene dato un accenno sulla tabella
(effettiva: 0,1,2,3...999,1000.. ossia il suo contenuto

UN ESEMPIO DI SINUS:

```
                          +          __
DEST>cosintabx            _ _ _/_ \_ _ _ _ _ _  = 512 words:
BEG>0                                    \__/
END>360                    -        0      360
AMOUNT>512        0,1,2,3...999,1000,999..3,2,0,-1,-2,-3..-1000,-999,...-2,-1,0
AMPLITUDE>1000
YOFFSET>0
SIZE (B/W/L)>W
MULTIPLIER>1
```

UN ESEMPIO DI COSINUS:

```
                           +           _          _
DEST>cosintabx             _ _ _ _\_ _ /_ _ _ _  = 512 words:
BEG>90                            \__/
END>360+90                   -        90      450
AMOUNT>512        1000,999..3,2,0,-1,-2,-3..-1000,-999,...-2,-1,0,1,2...999,1000
AMPLITUDE>1000
YOFFSET>0
SIZE (B/W/L)>W
MULTIPLIER>1
```

UN ALTRO ESEMPIO:

```
                          +          ___
DEST>cosintabx            _ _ _/_ _\_ _ _ _  = 800 words:
BEG>0
END>180                    -        0  180
AMOUNT>800                 0,1,2,3,4,5...999,1000,999..3,2,1,0 (800 valori)
AMPLITUDE>1000
YOFFSET>0
SIZE (B/W/L)>W
```

```
  MULTIPLIER>1


UN ALTRO ESEMPIO:                  _
                          +       / \
  DEST>cosintabx         _ _ _/_ _\_ _ _ _  = 800 words:
  BEG>0
  END>180                 -       0  180
  AMOUNT>800              0,1,2,3,4,5...1999,2000,1999..3,2,1,0 (800 valori)
  AMPLITUDE>1000
  YOFFSET>0
  SIZE (B/W/L)>W
  MULTIPLIER>2       <--


UN ALTRO ESEMPIO:                _       _
                          +         \    /
  DEST>cosintabx          _ _ _ _\__/_ _ _ _  = 512 words:
  BEG>90
  END>360+90                 -       90      450
  AMOUNT>512           2000,1999..3,2,0,1,2...1999,2000
  AMPLITUDE>1000
  YOFFSET>1000
  SIZE (B/W/L)>W
  MULTIPLIER>1


 ULTIMO ESEMPIO:                _       _
                          +         \    /
  DEST>cosintabx          _ _ _ _\__/_ _ _ _  = 360 words:
  BEG>90
  END>360+90                 -       90      450
  AMOUNT>360           304,303..3,2,0,1,2...303,304
  AMPLITUDE>152
  YOFFSET>152
  SIZE (B/W/L)>W
  MULTIPLIER>1
  HALF CORRECTION>Y
  ROUND CORRECTION>N

Ecco a voi come rifarsi le tabelle delle coordinate XX ed YY usate negli esempi
precedenti sugli sprite: (parametri per il CS e tabella finale)

Per le coordinate X, che devono andare da $40 a $d8 al massimo

; DEST> tabx
; BEG> 0                  ___ $d0
; END> 180                 /   \40
; AMOUNT> 200
; AMPLITUDE> $d0-$40        ; $40,$41,$42...$ce,$cf,d0,$cf,$ce...$43,$41....
; YOFFSET> $40          ; lo zero va trasformato in $40
; SIZE (B/W/L)> b
; MULTIPLIER> 1

        dc.b        $41,$43,$46,$48,$4A,$4C,$4F,$51,$53,$55,$58,$5A
        dc.b        $5C,$5E,$61,$63,$65,$67,$69,$6B,$6E,$70,$72,$74
        dc.b        $76,$78,$7A,$7C,$7E,$80,$82,$84,$86,$88,$8A,$8C
        dc.b        $8E,$90,$92,$94,$96,$97,$99,$9B,$9D,$9E,$A0,$A2
        dc.b        $A3,$A5,$A7,$A8,$AA,$AB,$AD,$AE,$B0,$B1,$B2,$B4
        dc.b        $B5,$B6,$B8,$B9,$BA,$BB,$BD,$BE,$BF,$C0,$C1,$C2
        dc.b        $C3,$C4,$C5,$C5,$C6,$C7,$C8,$C9,$C9,$CA,$CB,$CB
        dc.b        $CC,$CC,$CD,$CD,$CE,$CE,$CE,$CF,$CF,$CF,$CF,$D0
```

```
        dc.b        $D0,$D0,$D0,$D0,$D0,$D0,$D0,$D0,$CF,$CF,$CF
        dc.b        $CF,$CE,$CE,$CE,$CD,$CD,$CC,$CC,$CB,$CB,$CA,$C9
        dc.b        $C9,$C8,$C7,$C6,$C5,$C5,$C4,$C3,$C2,$C1,$C0,$BF
        dc.b        $BE,$BD,$BB,$BA,$B9,$B8,$B6,$B5,$B4,$B2,$B1,$B0
        dc.b        $AE,$AD,$AB,$AA,$A8,$A7,$A5,$A3,$A2,$A0,$9E,$9D
        dc.b        $9B,$99,$97,$96,$94,$92,$90,$8E,$8C,$8A,$88,$86
        dc.b        $84,$82,$80,$7E,$7C,$7A,$78,$76,$74,$72,$70,$6E
        dc.b        $6B,$69,$67,$65,$63,$61,$5E,$5C,$5A,$58,$55,$53
        dc.b        $51,$4F,$4C,$4A,$48,$46,$43,$41

--        --        --        --        --        --        --        --        --        --

; DEST> tabx                        $d0
; BEG> 180               \____/  $40
; END> 360
; AMOUNT> 200
; AMPLITUDE> $d0-$40        ; $cf,$cd,$ca...$42,$41,$40,$41,$42...$ca,$cd,$cf
; YOFFSET> $d0         ; curva sotto zero! allora bisogna aggiungere $d0
; SIZE (B/W/L)> b
; MULTIPLIER> 1

        dc.b        $CF,$CD,$CA,$C8,$C6,$C4,$C1,$BF,$BD,$BB,$B8,$B6
        dc.b        $B4,$B2,$AF,$AD,$AB,$A9,$A7,$A5,$A2,$A0,$9E,$9C
        dc.b        $9A,$98,$96,$94,$92,$90,$8E,$8C,$8A,$88,$86,$84
        dc.b        $82,$80,$7E,$7C,$7A,$79,$77,$75,$73,$72,$70,$6E
        dc.b        $6D,$6B,$69,$68,$66,$65,$63,$62,$60,$5F,$5E,$5C
        dc.b        $5B,$5A,$58,$57,$56,$55,$53,$52,$51,$50,$4F,$4E
        dc.b        $4D,$4C,$4B,$4B,$4A,$49,$48,$47,$47,$46,$45,$45
        dc.b        $44,$44,$43,$43,$42,$42,$42,$41,$41,$41,$41,$40
        dc.b        $40,$40,$40,$40,$40,$40,$40,$40,$40,$41,$41,$41
        dc.b        $41,$42,$42,$42,$43,$43,$44,$44,$45,$45,$46,$47
        dc.b        $47,$48,$49,$4A,$4B,$4B,$4C,$4D,$4E,$4F,$50,$51
        dc.b        $52,$53,$55,$56,$57,$58,$5A,$5B,$5C,$5E,$5F,$60
        dc.b        $62,$63,$65,$66,$68,$69,$6B,$6D,$6E,$70,$72,$73
        dc.b        $75,$77,$79,$7A,$7C,$7E,$80,$82,$84,$86,$88,$8A
        dc.b        $8C,$8E,$90,$92,$94,$96,$98,$9A,$9C,$9E,$A0,$A2
        dc.b        $A5,$A7,$A9,$AB,$AD,$AF,$B2,$B4,$B6,$B8,$BB,$BD
        dc.b        $BF,$C1,$C4,$C6,$C8,$CA,$CD,$CF

--        --        --        --        --        --        --        --        --        --

;                                  ___$d8
; DEST> tabx                         /   \ $d0-$40 ($90)
; BEG> 0                      \___/     $48
; END> 360
; AMOUNT> 200
; AMPLITUDE> ($d0-$40)/2 ; ampiezza sia sopra zero che sotto zero, allora
                         ; bisogna che faccia meta' sopra zero e meta' sotto,
                         ; ossia dividiamo per 2 l'AMPIEZZA
; YOFFSET> $90                ; e spostiamo tutto sopra per trasformare -72 in $48
; SIZE (B/W/L)> b
; MULTIPLIER> 1

        dc.b        $91,$93,$96,$98,$9A,$9C,$9F,$A1,$A3,$A5,$A7,$A9
        dc.b        $AC,$AE,$B0,$B2,$B4,$B6,$B8,$B9,$BB,$BD,$BF,$C0
        dc.b        $C2,$C4,$C5,$C7,$C8,$CA,$CB,$CC,$CD,$CF,$D0,$D1
        dc.b        $D2,$D3,$D3,$D4,$D5,$D5,$D6,$D7,$D7,$D7,$D8,$D8
        dc.b        $D8,$D8,$D8,$D8,$D8,$D8,$D7,$D7,$D7,$D6,$D5,$D5
        dc.b        $D4,$D3,$D3,$D2,$D1,$D0,$CF,$CD,$CC,$CB,$CA,$C8
        dc.b        $C7,$C5,$C4,$C2,$C0,$BF,$BD,$BB,$B9,$B8,$B6,$B4
        dc.b        $B2,$B0,$AE,$AC,$A9,$A7,$A5,$A3,$A1,$9F,$9C,$9A
        dc.b        $98,$96,$93,$91,$8F,$8D,$8A,$88,$86,$84,$81,$7F
```

```
        dc.b        $7D,$7B,$79,$77,$74,$72,$70,$6E,$6C,$6A,$68,$67
        dc.b        $65,$63,$61,$60,$5E,$5C,$5B,$59,$58,$56,$55,$54
        dc.b        $53,$51,$50,$4F,$4E,$4D,$4D,$4C,$4B,$4B,$4A,$49
        dc.b        $49,$49,$48,$48,$48,$48,$48,$48,$48,$48,$49,$49
        dc.b        $49,$4A,$4B,$4B,$4C,$4D,$4D,$4E,$4F,$50,$51,$53
        dc.b        $54,$55,$56,$58,$59,$5B,$5C,$5E,$60,$61,$63,$65
        dc.b        $67,$68,$6A,$6C,$6E,$70,$72,$74,$77,$79,$7B,$7D
        dc.b        $7F,$81,$84,$86,$88,$8A,$8D,$8F

--        --        --        --        --        --        --        --        --        --

 TABELLA DELLE Y:
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $2c e $f2, infatti nella tabella ci sono byte non
; piu' grandi di $f2 e non piu' piccoli di $2c.

; DEST> taby                          $f0 (d0)
; BEG> 180                \____/  $2c (40)
; END> 360
; AMOUNT> 200
; AMPLITUDE> $f0-$2c        ; $ef,$ed,$ea...$2c...$ea,$ed,$ef
; YOFFSET> $f0
; SIZE (B/W/L)> b
; MULTIPLIER> 1

        dc.b        $EE,$EB,$E8,$E5,$E2,$DF,$DC,$D9,$D6,$D3,$D0,$CD ; salto in
        dc.b        $CA,$C7,$C4,$C1,$BE,$BB,$B8,$B5,$B2,$AF,$AC,$A9 ; alto da
        dc.b        $A6,$A4,$A1,$9E,$9B,$98,$96,$93,$90,$8E,$8B,$88 ; record!
        dc.b        $86,$83,$81,$7E,$7C,$79,$77,$74,$72,$70,$6D,$6B
        dc.b        $69,$66,$64,$62,$60,$5E,$5C,$5A,$58,$56,$54,$52
        dc.b        $51,$4F,$4D,$4B,$4A,$48,$47,$45,$44,$42,$41,$3F
        dc.b        $3E,$3D,$3C,$3A,$39,$38,$37,$36,$35,$34,$33,$33
        dc.b        $32,$31,$30,$30,$2F,$2F,$2E,$2E,$2D,$2D,$2D,$2C
        dc.b        $2C,$2C,$2C,$2C,$2C,$2C,$2C,$2C,$2C,$2D,$2D,$2D
        dc.b        $2E,$2E,$2F,$2F,$30,$30,$31,$32,$33,$33,$34,$35
        dc.b        $36,$37,$38,$39,$3A,$3C,$3D,$3E,$3F,$41,$42,$44
        dc.b        $45,$47,$48,$4A,$4B,$4D,$4F,$51,$52,$54,$56,$58
        dc.b        $5A,$5C,$5E,$60,$62,$64,$66,$69,$6B,$6D,$70,$72
        dc.b        $74,$77,$79,$7C,$7E,$81,$83,$86,$88,$8B,$8E,$90
        dc.b        $93,$96,$98,$9B,$9E,$A1,$A4,$A6,$A9,$AC,$AF,$B2
        dc.b        $B5,$B8,$BB,$BE,$C1,$C4,$C7,$CA,$CD,$D0,$D3,$D6
        dc.b        $D9,$DC,$DF,$E2,$E5,$E8,$EB,$EE


--        --        --        --        --        --        --        --        --        --


;                                  ___ ($f0) $d8
; DEST> taby                         /   \ ($f0-$2c) $d0-$40 ($90)
; BEG> 0                      \___/      ($2c) $48
; END> 360
; AMOUNT> 200
; AMPLITUDE> ($f0-$2c)/2 ;
; YOFFSET> $8e                ; sarebbe $f0-(($f0-$2c)/2)
; SIZE (B/W/L)> b
; MULTIPLIER> 1

        dc.b        $8E,$91,$94,$97,$9A,$9D,$A0,$A3,$A6,$A9,$AC,$AF
        dc.b        $B2,$B4,$B7,$BA,$BD,$BF,$C2,$C5,$C7,$CA,$CC,$CE
        dc.b        $D1,$D3,$D5,$D7,$D9,$DB,$DD,$DF,$E0,$E2,$E3,$E5
        dc.b        $E6,$E7,$E9,$EA,$EB,$EC,$EC,$ED,$EE,$EE,$EF,$EF
        dc.b        $EF,$EF,$F0,$EF,$EF,$EF,$EF,$EE,$EE,$ED,$EC,$EC
```

```
        dc.b        $EB,$EA,$E9,$E7,$E6,$E5,$E3,$E2,$E0,$DF,$DD,$DB
        dc.b        $D9,$D7,$D5,$D3,$D1,$CE,$CC,$CA,$C7,$C5,$C2,$BF
        dc.b        $BD,$BA,$B7,$B4,$B2,$AF,$AC,$A9,$A6,$A3,$A0,$9D
        dc.b        $9A,$97,$94,$91,$8E,$8B,$88,$85,$82,$7F,$7C,$79
        dc.b        $76,$73,$70,$6D,$6A,$68,$65,$62,$5F,$5D,$5A,$57
        dc.b        $55,$52,$50,$4E,$4B,$49,$47,$45,$43,$41,$3F,$3D
        dc.b        $3C,$3A,$39,$37,$36,$35,$33,$32,$31,$30,$30,$2F
        dc.b        $2E,$2E,$2D,$2D,$2D,$2D,$2C,$2D,$2D,$2D,$2D,$2E
        dc.b        $2E,$2F,$30,$30,$31,$32,$33,$35,$36,$37,$39,$3A
        dc.b        $3C,$3D,$3F,$41,$43,$45,$47,$49,$4B,$4E,$50,$52
        dc.b        $55,$57,$5A,$5D,$5F,$62,$65,$68,$6A,$6D,$70,$73
        dc.b        $76,$79,$7C,$7F,$82,$85,$88,$8B,$8d

--        --        --        --        --        --        --        --        --        --

Dato che avete tutte queste tabelle XX ed YY pronte provate a sostituirle a
quelle del listato, per creare molti effetti diversi, e provate a farne altre
con 100, 120, 300 valori anziche' 200 ( AMOUNT> 100), per creare infinite
traiettorie dello sprite.
```

23.6 Lezione7f

```
; Lezione7f.s        VISUALIZZAZIONE DI TUTTI GLI 8 SPRITE DELL'AMIGA
;                In questo listato viene verificato che gli 8 sprite hanno
;                la palette in comune a coppie, ossia lo sprite 0 ha gli stessi
;                colori dello sprite 1, lo sprite 2 ha gli stessi dello sprite 3
;                e cosi' via. Viene verificato anche che nel caso della
;                sovrapposizione di due sprite, quello con numero minore
;                prevale su quello con numero maggiore, per cui lo sprite 0
;                appare sopra tutti gli altri e lo sprite 7 puo' essere coperto
;                da tutti gli altri, mentre lo sprite 3 copre gli sprite 4,5,6,7
;                ed e' coperto dagli sprite 0,1,2
;                Premendo il tasto sinistro gli sprite si sovrappongono e si
;                notano le priorita' di sovrapposizione. Tasto destro del mouse
;                per uscire.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo gli sprite

        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
```

```
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE1,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE2,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE3,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE4,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE5,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE6,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE7,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        MOVEQ        #$60,d0                ; Coordinata HSTART iniziale
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
                                ; (da notare che il byte HSTART lavora sui
                                ; pixel a 2 a 2, per cui per spostarsi di 10
                                ; pixel basta aggiungere 5 ad HSTART!
        MOVE.B        d0,HSTART1
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
        MOVE.B        d0,HSTART2
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
        MOVE.B        d0,HSTART3
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
        MOVE.B        d0,HSTART4
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
        MOVE.B        d0,HSTART5
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
```

```
        MOVE.B        d0,HSTART6
        ADDQ.B        #(10/2),d0        ; distanza col prossimo sprite
        MOVE.B        d0,HSTART7

MouseDestro:
        btst        #2,$dff016
        bne.s        MouseDestro

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, - COLOR1 degli sprite0/1 -ROSSO
        dc.w        $1A4,$0F0        ; color18, - COLOR2 degli sprite0/1 -VERDE
        dc.w        $1A6,$FF0        ; color19, - COLOR3 degli sprite0/1 -GIALLO
```

```
        dc.w        $1AA,$FFF        ; color21, - COLOR1 degli sprite2/3 -BIANCO
        dc.w        $1AC,$0BD        ; color22, - COLOR2 degli sprite2/3 -ACQUA
        dc.w        $1AE,$D50        ; color23, - COLOR3 degli sprite2/3 -ARANCIO

        dc.w        $1B2,$00F        ; color25, - COLOR1 degli sprite4/5 -BLU
        dc.w        $1B4,$F0F        ; color26, - COLOR2 degli sprite4/5 -VIOLA
        dc.w        $1B6,$BBB        ; color27, - COLOR3 degli sprite4/5 -GRIGIO

        dc.w        $1BA,$8E0        ; color29, - COLOR1 degli sprite6/7 -VERDE CH.
        dc.w        $1BC,$a70        ; color30, - COLOR2 degli sprite6/7 -MARRONE
        dc.w        $1BE,$d00        ; color31, - COLOR3 degli sprite6/7 -ROSSO SC.

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! ************

 ; tabella di riferimento per definire i colori:


 ; per gli sprite 0 ed 1
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (ROSSO)
 ;BINARIO 01=COLORE 2 (VERDE)
 ;BINARIO 11=COLORE 3 (GIALLO)

MIOSPRITE0:                ; lunghezza 13 linee
VSTART0:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART0:
        dc.b $60        ; Pos. orizzontale (da $40 a $d8)
VSTOP0:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite


MIOSPRITE1:                ; lunghezza 13 linee
VSTART1:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART1:
        dc.b $60+14        ; Pos. orizzontale (da $40 a $d8)
VSTOP1:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000010001111
 dc.w        %0011111111111100,%1100000110000011
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0011111111111100,%1100000010000011
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite
```

```
 ; per gli sprite 2 e 3
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BIANCO)
 ;BINARIO 01=COLORE 2 (ACQUA)
 ;BINARIO 11=COLORE 3 (ARANCIO)

MIOSPRITE2:                ; lunghezza 13 linee
VSTART2:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART2:
        dc.b $60+(14*2)        ; Pos. orizzontale (da $40 a $d8)
VSTOP2:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0011111111111100,%1100001000100011
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0111111111111110,%1000000111000001
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

MIOSPRITE3:                ; lunghezza 13 linee
VSTART3:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART3:
        dc.b $60+(14*3)        ; Pos. orizzontale (da $40 a $d8)
VSTOP3:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0111111111111110,%1000000111100001
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

 ; per gli sprite 4 e 5
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BLU)
 ;BINARIO 01=COLORE 2 (VIOLA)
 ;BINARIO 11=COLORE 3 (GRIGIO)

MIOSPRITE4:                ; lunghezza 13 linee
VSTART4:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART4:
        dc.b $60+(14*4)        ; Pos. orizzontale (da $40 a $d8)
VSTOP4:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001001001111
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
```

```
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

MIOSPRITE5:                 ; lunghezza 13 linee
VSTART5:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART5:
        dc.b $60+(14*5)         ; Pos. orizzontale (da $40 a $d8)
VSTOP5:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

 ; per gli sprite 6 e 7
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (VERDE CHIARO)
 ;BINARIO 01=COLORE 2 (MARRONE)
 ;BINARIO 11=COLORE 3 (ROSSO SCURO)

MIOSPRITE6:                 ; lunghezza 13 linee
VSTART6:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART6:
        dc.b $60+(14*6)         ; Pos. orizzontale (da $40 a $d8)
VSTOP6:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000001001000001
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

MIOSPRITE7:                 ; lunghezza 13 linee
VSTART7:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART7:
        dc.b $60+(14*7)         ; Pos. orizzontale (da $40 a $d8)
VSTOP7:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
```

```
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo listato vengono "puntati" tutti gli 8 sprites, i quali hanno il numero nel disegno per rendere piu' chiara la loro disposizione.
Come spiegato nella teoria, gli 8 sprite hanno 4 palette distinte di colori, per cui gli sprite adiacenti condivisono la stessa palette:

```
        dc.w        $1A2,$F00        ; color17, - COLOR1 degli sprite0/1 -ROSSO
        dc.w        $1A4,$0F0        ; color18, - COLOR2 degli sprite0/1 -VERDE
        dc.w        $1A6,$FF0        ; color19, - COLOR3 degli sprite0/1 -GIALLO

        dc.w        $1AA,$FFF        ; color21, - COLOR1 degli sprite2/3 -BIANCO
        dc.w        $1AC,$0BD        ; color22, - COLOR2 degli sprite2/3 -ACQUA
        dc.w        $1AE,$D50        ; color23, - COLOR3 degli sprite2/3 -ARANCIO

        dc.w        $1B2,$00F        ; color25, - COLOR1 degli sprite4/5 -BLU
        dc.w        $1B4,$F0F        ; color26, - COLOR2 degli sprite4/5 -VIOLA
        dc.w        $1B6,$BBB        ; color27, - COLOR3 degli sprite4/5 -GRIGIO

        dc.w        $1BA,$8E0        ; color29, - COLOR1 degli sprite6/7 -VERDE CH.
        dc.w        $1BC,$a70        ; color30, - COLOR2 degli sprite6/7 -MARRONE
        dc.w        $1BE,$d00        ; color31, - COLOR3 degli sprite6/7 -ROSSO SC.
```

Da notare che i colori Color16,Color20,Color24 e Color28 non sono usati dagli sprite, vengono saltati, in quanto corrisponderebbero al color0 dello sprite, quello TRASPARENTE, che non e', appunto, un colore, ma un "BUCO" che assume il colore dei bitplane (o sprites) sottostanti.
Ogni sprite ha il suo VSTART,HSTART e VSTOP, vediamo ad esempio lo SPRITE2:

```
MIOSPRITE2:                ; lunghezza 13 linee
VSTART2:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART2:
        dc.b $60+(14*2)        ; Pos. orizzontale (da $40 a $d8)
VSTOP2:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
```

Ogni sprite all'inizio e' distanziato dagli altri, tramite l'aggiunta di (14*x) agli HSTART. Dopo la pressione del tasto sinistro del mouse vengono cambiati tutti gli HSTART meno il primo in modo da sovrapporre gli sprite e mostrare le priorita' di visualizzazione tra di essi.

23.7 Lezione7g

```
; Lezione7g.s        UNO SPRITE A 16 COLORI IN MODO ATTACCHED MOSSO SULLO SCHERMO
;                 USANDO DUE TABELLE DI VALORI (ossia di coordinate verticali
;                e orizzontali) PRESTABILITI.
```

```
        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)         ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0         ; dove puntare
        LEA         BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

;        Puntiamo gli sprite 0 ed 1, che ATTACCATI formeranno un solo sprite
;        a 16 colori. Lo sprite1, quello dispari, deve avere il bit 7 della
;        seconda word ad 1.

        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE1,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        bset         #7,MIOSPRITE1+3                ; Setta il bit dell'attacched allo
                                        ; sprite 1. Togliendo questa istruzione
                                        ; gli sprite non sono ATTACCHED, ma
                                        ; due a 3 colori sovrapposti.

        move.l        #COPPERLIST,$dff080         ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006         ; Linea 255?
        bne.s         mouse

        bsr.s         MuoviSpriteX         ; Muovi lo sprite 0 orizzontalmente
        bsr.w         MuoviSpriteY         ; Muovi lo sprite 0 verticalmente

Aspetta:
        cmpi.b        #$ff,$dff006         ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001         ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080         ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
```

```
        jsr        -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine sposta  lo sprite agendo sul suo byte HSTART, ossia
; il byte della sua posizione X, immettendoci delle coordinate gia' stabilite
; nella tabella TABX. (Scatti di 2 pixel alla volta)

MuoviSpriteX:
        ADDQ.L        #1,TABXPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT ; Riparti a puntare dal primo byte-1
NOBSTARTX:
        MOVE.b        (A0),MIOSPRITE0+1 ; copia il byte dalla tabella ad HSTART0
        MOVE.b        (A0),MIOSPRITE1+1 ; copia il byte dalla tabella ad HSTART1
        rts

TABXPOINT:
        dc.l        TABX-1               ; NOTA: i valori della tabella sono bytes

; Tabella con coordinate X dello sprite precalcolate.

TABX:
        incbin        "XCOORDINAT.TAB"        ; 334 valori
FINETABX:


; Questa routine sposta in alto e in basso lo sprite agendo sui suoi byte
; VSTART e VSTOP, ossia i byte della sua posizione Y di inizio e fine,
; immettendoci delle coordinate gia' stabilite nella tabella TABY

MuoviSpriteY:
        ADDQ.L        #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte (-1)
NOBSTARTY:
        moveq        #0,d0               ; Pulisci d0
        MOVE.b        (A0),d0               ; copia il byte dalla tabella in d0
        MOVE.b        d0,MIOSPRITE0        ; copia il byte in VSTART0
        MOVE.b        d0,MIOSPRITE1        ; copia il byte in VSTART1
        ADD.B        #15,D0               ; Aggiungi la lunghezza dello sprite per
                             ; determinare la posizione finale (VSTOP)
        move.b        d0,MIOSPRITE0+2        ; Muovi il valore giusto in VSTOP0
        move.b        d0,MIOSPRITE1+2        ; Muovi il valore giusto in VSTOP1
```

```
        rts

TABYPOINT:
        dc.l        TABY-1              ; NOTA: i valori della tabella sono bytes

; Tabella con coordinate Y dello sprite precalcolate.

TABY:
        incbin        "YCOORDINAT.TAB"        ; 200 valori
FINETABY:


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

;        Palette della PIC

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

;        Palette degli SPRITE attacched

        dc.w        $1A2,$FFC        ; color17, COLORE 1 per gli sprite attaccati
        dc.w        $1A4,$EEB        ; color18, COLORE 2 per gli sprite attaccati
        dc.w        $1A6,$CD9        ; color19, COLORE 3 per gli sprite attaccati
        dc.w        $1A8,$AC8        ; color20, COLORE 4 per gli sprite attaccati
        dc.w        $1AA,$8B6        ; color21, COLORE 5 per gli sprite attaccati
        dc.w        $1AC,$6A5        ; color22, COLORE 6 per gli sprite attaccati
        dc.w        $1AE,$494        ; color23, COLORE 7 per gli sprite attaccati
        dc.w        $1B0,$384        ; color24, COLORE 7 per gli sprite attaccati
        dc.w        $1B2,$274        ; color25, COLORE 9 per gli sprite attaccati
        dc.w        $1B4,$164        ; color26, COLORE 10 per gli sprite attaccati
        dc.w        $1B6,$154        ; color27, COLORE 11 per gli sprite attaccati
        dc.w        $1B8,$044        ; color28, COLORE 12 per gli sprite attaccati
        dc.w        $1BA,$033        ; color29, COLORE 13 per gli sprite attaccati
        dc.w        $1BC,$012        ; color30, COLORE 14 per gli sprite attaccati
        dc.w        $1BE,$001        ; color31, COLORE 15 per gli sprite attaccati

        dc.w        $FFFF,$FFFE        ; Fine della copperlist
```

```
; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! **********

MIOSPRITE0:                ; lunghezza 15 linee
VSTART0:
        dc.b $00        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART0:
        dc.b $00        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP0:
        dc.b $00        ; posizione verticale di fine sprite
        dc.b $00

        dc.w $0380,$0650,$04e8,$07d0,$0534,$1868,$1e5c,$1636 ; dati dello
        dc.w $377e,$5514,$43a1,$1595,$0172,$1317,$6858,$5035 ; sprite 0
        dc.w $318c,$0c65,$7453,$27c9,$5ece,$5298,$0bfe,$2c32
        dc.w $005c,$13c4,$0be8,$0c18,$03e0,$03e0

        dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.



MIOSPRITE1:                ; lunghezza 15 linee
VSTART1:
        dc.b $00        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART1:
        dc.b $00        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP1:
        dc.b $00        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00        ; settare il bit 7 per attaccare sprite 0 ed 1.

        dc.w $0430,$07f0,$0fc8,$0838,$0fe4,$101c,$39f2,$200e ; dati dello
        dc.w $58f2,$600e,$5873,$600f,$5cf1,$600f,$1ff3,$600f ; sprite 1
        dc.w $4fe3,$701f,$47c7,$783f,$6286,$7d7e,$300e,$3ffe
        dc.w $1c3c,$1ffc,$0ff8,$0ff8,$03e0,$03e0

        dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

A parte la novita' del bit ATTACCHED per fare uno sprite a 16 colori anziche' due a 4 colori, sono da notare un paio di cose:
1) Le tabelle X ed Y sono state salvate col comando "WB" e vengono caricate con l'incbin, in questo modo le tabelle possono essere caricate dai vari listati che le richiedono, basta che siano sul disco!
2) Non vengono piu' usate le label VSTART0,VSTART1,HSTART0,HSTART1 ecc. per muovere lo sprite. Le label rimangono al loro posto nello sprite in questo listato, ma risulta piu' comodo "raggiungere" i byte di controllo cosi':

```
        MIOSPRITE        ; Per VSTART
        MIOSPRITE+1        ; Per HSTART
        MIOSPRITE+2        ; Per VSTOP
```

In questo modo si puo' semplicemente cominciare lo sprite con:

```
MIOSPRITE:
        DC.W        0,0
        ..dati...
```

Senza dividere le due word in singoli byte, ognuno con una LABEL che allunga il listato.
Anche per settare il bit 7 della word 2 dello SPRITE1, quello dell'ATTACCHED, e' bastata questa istruzione:

```
        bset        #7,MIOSPRITE1+3
```

Altrimento avremmo potuto settarlo "a mano" nel quarto byte:

```
MIOSPRITE1:
VSTART1:
        dc.b $00
HSTART1:
        dc.b $00
VSTOP1:
        dc.b $00
        dc.b %10000000              ; oppure dc.b $80 ($80=%10000000)
```

Se si devono usare tutti e 8 gli sprite si risparmiano un bel po' di label e di spazio. Ancora meglio sarebbe mettere in un registro Ax l'indirizzo dello sprite ed eseguire gli offset da quel registro:

```
        lea        MIOSPRITE,a0
        MOVE.B        #yy,(a0)        ; Per VSTART
        MOVE.B        #xx,1(A0)        ; Per HSTART
        MOVE.B        #y2,2(A0)        ; Per VSTOP
```

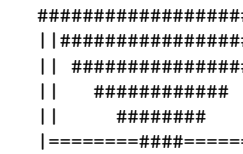
Definirsi in binario uno sprite a 16 colori diventa problematico.
Dunque bisogna ricorrere ad un programma di disegno, basta ricordarsi di usare uno schermo a 16 colori e di disegnare gli sprite non piu' larghi di 16 pixel. Una volta salvata la PIC a 16 colori (o un BRUSH piu' piccolo con lo sprite) in formato IFF, convertirlo con l'IFFCONVERTER e' facile come convertire una figura.

NOTA: Per BRUSH si intende un pezzo di figura di dimensioni variabili.

Ecco come potete convertirvi uno sprite col KEFCON:

1) Caricate il file IFF, che deve essere a 16 colori
2) Dovete selezionare solo lo sprite, per fare cio' premete il tasto destro, poi posizionatevi sull'angolo in alto a sinistra del futuro sprite, e premete il tasto sinistro. Muovendo il mouse vi apparira' una griglia che, guarda caso, e' divisa a strisce larghe 16 pixel. Potete comunque controllare la larghezza e la lunghezza del blocco selezionato. Per includere bene lo sprite dovete considerare che dovete passare per il bordo dello sprite con la "striscia" di selezione del rettangolo, l'ultima linea inclusa nel rettangolo e' quella che passa per la striscia di confine, non e' quella interna alla striscia:

```
        <----- 16 pixel ----->

        |=======####========| /\
        ||     ########            || ||
        ||   #############   || ||
        || ################## || ||
        ||###################|| ||
        ######################## ||
        ######################## Lunghezza dello sprite, massimo 256 pixel
```



Se lo sprite e' piu' piccolo di 16 pixel dovete lasciare un margine vuoto ai lati, o ad un solo lato, in modo che la larghezza del blocco sia sempre 16.

Una volta selezionato lo sprite dentro il rettangolo, occorre salvarlo come SPRITE16 se e' uno sprite a 16 colori, o come SPRITE4 se e' uno sprite a quattro colori. Lo sprite viene salvato in "dc.b", ossia in formato TESTO, che potete includere nel listato col comando "I" dell'Asmone o caricandolo in un altro buffer di testo e copiandolo con Amiga+b+c+i.

Ecco come il KEFCON salva lo sprite attacched (16 colori):

```
        dc.w $0000,$0000
        dc.w $0380,$0650,$04e8,$07d0,$0534,$1868,$1e5c,$1636
        dc.w $377e,$5514,$43a1,$1595,$0172,$1317,$6858,$5035
        dc.w $318c,$0c65,$7453,$27c9,$5ece,$5298,$0bfe,$2c32
        dc.w $005c,$13c4,$0be8,$0c18,$03e0,$03e0
        dc.w 0,0

        dc.w $0000,$0000
        dc.w $0430,$07f0,$0fc8,$0838,$0fe4,$101c,$39f2,$200e
        dc.w $58f2,$600e,$5873,$600f,$5cf1,$600f,$1ff3,$600f
        dc.w $4fe3,$701f,$47c7,$783f,$6286,$7d7e,$300e,$3ffe
        dc.w $1c3c,$1ffc,$0ff8,$0ff8,$03e0,$03e0
        dc.w 0,0
```

Come potete notare, questi sono i due sprite con le due word di controllo azzerate, i dati in formato esadecimale e le due word azzerate di FINE SPRITE. Basta mettere le due label "MIOSPRITE0:" e "MIOSPRITE1:" all'inizio dei due sprite, dopodiche' lavorando con MIOSPRITE+x per raggiungere i byte delle coordinate non occorre aggiungere altre LABEL. L'unico particolare e' che bisogna settare il bit dell'ATTACCHED con un BSET #7,MIOSPRITE+3 oppure direttamente nello sprite:

```
MIOSPRITE1:
        dc.w $0000,$0080        ; $80, ossia %10000000 -> ATTACCHED!
        dc.w $0430,$07f0,$0fc8,$0838,$0fe4,$101c,$39f2,$200e
        ...
```

Se volete disegnarvi e convertirvi anche gli sprite a 4 colori, il problema non sussiste, perche' viene salvato un solo sprite e non occorre settare il bit!

Per quanto riguarda la palette dei colori degli sprite, bisogna salvarli dal KEFCON dopo aver salvato lo SPRITE16 o SPRITE4, con l'opzione COPPER, proprio come per le figure normali. Il problemuccio e' che viene salvata la palette intesa come FIGURA a 16 COLORI, e non come SPRITE.
Ecco come il KEFCON salva la palette:

```
        dc.w $0180,$0000,$0182,$0ffc,$0184,$0eeb,$0186,$0cd9
        dc.w $0188,$0ac8,$018a,$08b6,$018c,$06a5,$018e,$0494
        dc.w $0190,$0384,$0192,$0274,$0194,$0164,$0196,$0154
        dc.w $0198,$0044,$019a,$0033,$019c,$0012,$019e,$0001
```

I colori sono giusti, ma i registri colore si riferiscono ai primi 16 colori e non gli ultimi 16. Basta riscriverli "a mano" nei registri colore giusti:

```
        dc.w        $1A2,$FFC       ; color17, COLORE 1 per gli sprite attaccati
        dc.w        $1A4,$EEB       ; color18, COLORE 2 per gli sprite attaccati
        dc.w        $1A6,$CD9       ; color19, COLORE 3 per gli sprite attaccati
        dc.w        $1A8,$AC8       ; color20, COLORE 4 per gli sprite attaccati
        dc.w        $1AA,$8B6       ; color21, COLORE 5 per gli sprite attaccati
        dc.w        $1AC,$6A5       ; color22, COLORE 6 per gli sprite attaccati
        dc.w        $1AE,$494       ; color23, COLORE 7 per gli sprite attaccati
        dc.w        $1B0,$384       ; color24, COLORE 7 per gli sprite attaccati
        dc.w        $1B2,$274       ; color25, COLORE 9 per gli sprite attaccati
        dc.w        $1B4,$164       ; color26, COLORE 10 per gli sprite attaccati
        dc.w        $1B6,$154       ; color27, COLORE 11 per gli sprite attaccati
        dc.w        $1B8,$044       ; color28, COLORE 12 per gli sprite attaccati
        dc.w        $1BA,$033       ; color29, COLORE 13 per gli sprite attaccati
        dc.w        $1BC,$012       ; color30, COLORE 14 per gli sprite attaccati
        dc.w        $1BE,$001       ; color31, COLORE 15 per gli sprite attaccati

Si noti che in $1a2 bisogna copiare il colore in $182, in $1a4 quello in $184
e cosi' via.

Provate a sostituire lo sprite a 16 colori di questo listato con uno vostro,
con la vostra palette colori, e anche a convertirne uno a 4 colori da
sostituire a quello delle lezioni precedenti. Farlo servira' da verifica!!!
```

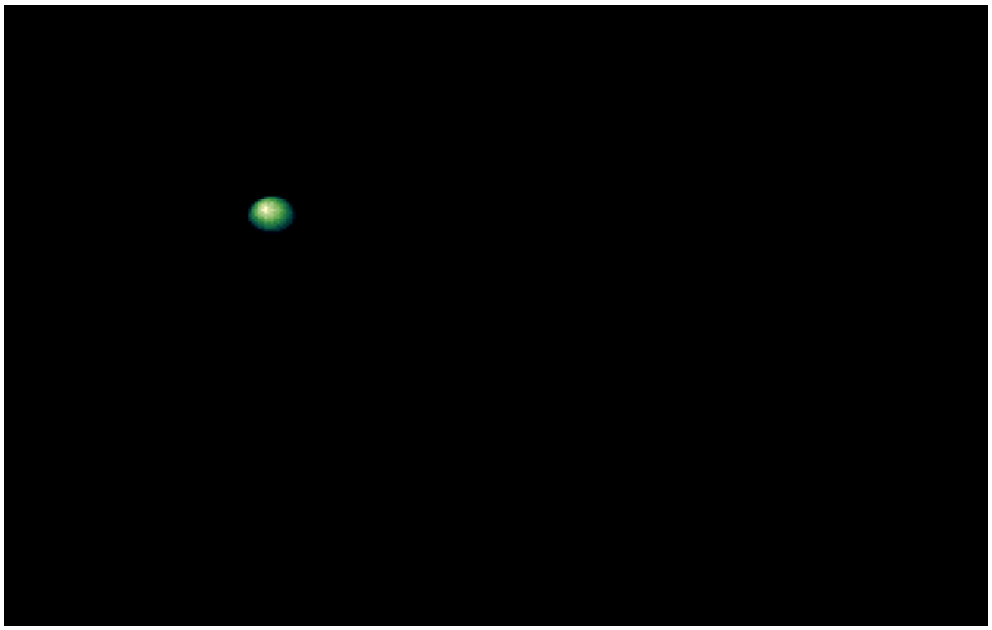


Figura 23.2: Lezione 7g

23.8 Lezione7h

```
; Lezione7h.s        4 SPRITE A 16 COLORI IN MODO ATTACCHED MOSSI SULLO SCHERMO
;                  USANDO DUE TABELLE DI VALORI (ossia di coordinate verticali
;                 e orizzontali) PRESTABILITI.
```

```
;               ** NOTA ** Per vedere il programma ed uscire premere:
;               TASTO SINISTRO, TASTO DESTRO, TASTO SINISTRO, TASTO DESTRO.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)         ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0         ; dove puntare
        LEA         BPLPOINTERS,A1         ; puntatori COP
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

;        Puntiamo gli 8 sprite, che ATTACCATI formeranno 4 sprite a 16 colori.
;        Gli sprite 1,3,5,7, quelli dispari, devono avere il bit 7 della
;        seconda word ad 1.


        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1         ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE1,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE2,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE3,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE4,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE5,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE6,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
```

```
        MOVE.L        #MIOSPRITE7,d0               ; indirizzo dello sprite in d0
        addq.w        #8,a1                       ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

; Settiamo i bit dell'ATTACCHED

        bset        #7,MIOSPRITE1+3               ; Setta il bit dell'attacched allo
                                    ; sprite 1. Togliendo questa istruzione
                                    ; gli sprite non sono ATTACCHED, ma
                                    ; due a 3 colori sovrapposti.

        bset        #7,MIOSPRITE3+3
        bset        #7,MIOSPRITE5+3
        bset        #7,MIOSPRITE7+3

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

;        Creiamo una differenza di posizione nei puntatori alle tabelle tra i
;        4 sprite per fargli fare movimenti diversi l'uno dall'altro.

        MOVE.L        #TABX+55,TABXPOINT0
        MOVE.L        #TABX+86,TABXPOINT1
        MOVE.L        #TABX+130,TABXPOINT2
        MOVE.L        #TABX+170,TABXPOINT3
        MOVE.L        #TABY-1,TABYPOINT0
        MOVE.L        #TABY+45,TABYPOINT1
        MOVE.L        #TABY+90,TABYPOINT2
        MOVE.L        #TABY+140,TABYPOINT3


Mouse1:
        bsr.w        MuoviGliSprite        ; Attende un fotogramma, muove gli sprite e
                            ; ritorna.

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse1

        MOVE.L        #TABX+170,TABXPOINT0
        MOVE.L        #TABX+130,TABXPOINT1
        MOVE.L        #TABX+86,TABXPOINT2
        MOVE.L        #TABX+55,TABXPOINT3
        MOVE.L        #TABY-1,TABYPOINT0
        MOVE.L        #TABY+45,TABYPOINT1
        MOVE.L        #TABY+90,TABYPOINT2
        MOVE.L        #TABY+140,TABYPOINT3

Mouse2:
        bsr.w        MuoviGliSprite        ; Attende un fotogramma, muove gli sprite e
                            ; ritorna.

        btst        #2,$dff016        ; tasto destro del mouse premuto?
        bne.s        mouse2

; SPRITE IN FILA INDIANA

        MOVE.L        #TABX+30,TABXPOINT0
        MOVE.L        #TABX+20,TABXPOINT1
```

```
        MOVE.L        #TABX+10,TABXPOINT2
        MOVE.L        #TABX-1,TABXPOINT3
        MOVE.L        #TABY+30,TABYPOINT0
        MOVE.L        #TABY+20,TABYPOINT1
        MOVE.L        #TABY+10,TABYPOINT2
        MOVE.L        #TABY-1,TABYPOINT3

Mouse3:
        bsr.w         MuoviGliSprite        ; Attende un fotogramma, muove gli sprite e
                                            ; ritorna.

        btst         #6,$bfe001             ; tasto sinistro del mouse premuto?
        bne.s         mouse3

; SPRITE UBRIACHI PER LO SCHERMO

        MOVE.L        #TABX+220,TABXPOINT0
        MOVE.L        #TABX+30,TABXPOINT1
        MOVE.L        #TABX+102,TABXPOINT2
        MOVE.L        #TABX+5,TABXPOINT3
        MOVE.L        #TABY-1,TABYPOINT0
        MOVE.L        #TABY+180,TABYPOINT1
        MOVE.L        #TABY+20,TABYPOINT2
        MOVE.L        #TABY+100,TABYPOINT3


Mouse4:
        bsr.w         MuoviGliSprite        ; Attende un fotogramma, muove gli sprite e
                                            ; ritorna.

        btst         #2,$dff016             ; tasto destro del mouse premuto?
        bne.s         mouse4

        move.l         OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w         d0,$dff088                ; facciamo partire la vecchia cop


        move.l         4.w,a6
        jsr           -$7e(a6)         ; Enable
        move.l         gfxbase(PC),a1
        jsr           -$19e(a6)         ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b          "graphics.library",0,0

GfxBase:
        dc.l          0

OldCop:
        dc.l          0


; Questa routine esegue le singole routines di movimento degli sprite
; ed include anche il loop di attesa del fotogramma per la temporizzazione.

MuoviGliSprite:
        cmpi.b         #$ff,$dff006         ; Linea 255?
        bne.s         MuoviGliSprite
```

```
        bsr.s        MuoviSpriteX0        ; Muovi lo sprite 0 orizzontalmente
        bsr.w        MuoviSpriteX1        ; Muovi lo sprite 1 orizzontalmente
        bsr.w        MuoviSpriteX2        ; Muovi lo sprite 2 orizzontalmente
        bsr.w        MuoviSpriteX3        ; Muovi lo sprite 3 orizzontalmente
        bsr.w        MuoviSpriteY0        ; Muovi lo sprite 0 verticalmente
        bsr.w        MuoviSpriteY1        ; Muovi lo sprite 1 verticalmente
        bsr.w        MuoviSpriteY2        ; Muovi lo sprite 2 verticalmente
        bsr.w        MuoviSpriteY3        ; Muovi lo sprite 3 verticalmente

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        rts                ; Torna al loop MOUSE


; ******************** ROUTINES DI MOVIMENTO ORIZZONTALE ******************

; Queste routines spostano lo sprite agendo sul suo byte HSTART, ossia
; il byte della sua posizione X, immettendoci delle coordinate gia' stabilite
; nella tabella TABX (Scorrimento orizzontale a scatti di 2 pixel e non 1)

; Per lo sprite0 ATTACCHED: (ossia Sprite0+Sprite1)

MuoviSpriteX0:
        ADDQ.L        #1,TABXPOINT0         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT0(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTX0        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT0 ; Riparti a puntare dal primo byte-1
NOBSTARTX0:
        MOVE.b        (A0),MIOSPRITE0+1 ; copia il byte dalla tabella ad HSTART0
        MOVE.b        (A0),MIOSPRITE1+1 ; copia il byte dalla tabella ad HSTART1
        rts

TABXPOINT0:
        dc.l        TABX+55                ; NOTA: i valori della tabella sono bytes



; Per lo sprite1 ATTACCHED: (ossia Sprite2+Sprite3)

MuoviSpriteX1:
        ADDQ.L        #1,TABXPOINT1         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT1(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTX1        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT1 ; Riparti a puntare dal primo byte-1
NOBSTARTX1:
        MOVE.b        (A0),MIOSPRITE2+1 ; copia il byte dalla tabella ad HSTART2
        MOVE.b        (A0),MIOSPRITE3+1 ; copia il byte dalla tabella ad HSTART3
        rts

TABXPOINT1:
        dc.l        TABX+86                ; NOTA: i valori della tabella sono bytes



; Per lo sprite2 ATTACCHED: (ossia Sprite4+Sprite5)
```

```
MuoviSpriteX2:
        ADDQ.L        #1,TABXPOINT2         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT2(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTX2        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT2 ; Riparti a puntare dal primo byte-1
NOBSTARTX2:
        MOVE.b        (A0),MIOSPRITE4+1 ; copia il byte dalla tabella ad HSTART4
        MOVE.b        (A0),MIOSPRITE5+1 ; copia il byte dalla tabella ad HSTART5
        rts

TABXPOINT2:
        dc.l        TABX+130        ; NOTA: i valori della tabella sono bytes



; Per lo sprite3 ATTACCHED: (ossia Sprite6+Sprite7)

MuoviSpriteX3:
        ADDQ.L        #1,TABXPOINT3         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT3(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABX-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTX3        ; non ancora? allora continua
        MOVE.L        #TABX-1,TABXPOINT3 ; Riparti a puntare dal primo byte-1
NOBSTARTX3:
        MOVE.b        (A0),MIOSPRITE6+1 ; copia il byte dalla tabella ad HSTART6
        MOVE.b        (A0),MIOSPRITE7+1 ; copia il byte dalla tabella ad HSTART7
        rts

TABXPOINT3:
        dc.l        TABX+170        ; NOTA: i valori della tabella sono bytes

; ********************* ROUTINES DI MOVIMENTO VERTICALE ******************

; Queste routines spostano in alto e in basso lo sprite agendo sui suoi byte
; VSTART e VSTOP, ossia i byte della sua posizione Y di inizio e fine,
; immettendoci delle coordinate gia' stabilite nella tabella TABY

; Per lo sprite0 ATTACCHED: (ossia Sprite0+Sprite1)

MuoviSpriteY0:
        ADDQ.L        #1,TABYPOINT0         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT0(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTY0        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT0 ; Riparti a puntare dal primo byte (-1)
NOBSTARTY0:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte dalla tabella in d0
        MOVE.b        d0,MIOSPRITE0        ; copia il byte in VSTART0
        MOVE.b        d0,MIOSPRITE1        ; copia il byte in VSTART1
        ADD.B        #15,D0                ; Aggiungi la lunghezza dello sprite per
                              ; determinare la posizione finale (VSTOP)
        move.b        d0,MIOSPRITE0+2        ; Muovi il valore giusto in VSTOP0
        move.b        d0,MIOSPRITE1+2        ; Muovi il valore giusto in VSTOP1
        rts

TABYPOINT0:
        dc.l        TABY-1                ; NOTA: i valori della tabella sono bytes
```

```
; Per lo sprite1 ATTACCHED: (ossia Sprite2+Sprite3)

MuoviSpriteY1:
        ADDQ.L        #1,TABYPOINT1         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT1(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTY1        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT1 ; Riparti a puntare dal primo byte (-1)
NOBSTARTY1:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte dalla tabella in d0
        MOVE.b        d0,MIOSPRITE2        ; copia il byte in VSTART2
        MOVE.b        d0,MIOSPRITE3        ; copia il byte in VSTART3
        ADD.B        #15,D0                ; Aggiungi la lunghezza dello sprite per
                           ; determinare la posizione finale (VSTOP)
        move.b        d0,MIOSPRITE2+2        ; Muovi il valore giusto in VSTOP2
        move.b        d0,MIOSPRITE3+2        ; Muovi il valore giusto in VSTOP3
        rts

TABYPOINT1:
        dc.l        TABY+45                ; NOTA: i valori della tabella sono bytes


; Per lo sprite2 ATTACCHED: (ossia Sprite4+Sprite5)

MuoviSpriteY2:
        ADDQ.L        #1,TABYPOINT2         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT2(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTY2        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT2 ; Riparti a puntare dal primo byte (-1)
NOBSTARTY2:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte dalla tabella in d0
        MOVE.b        d0,MIOSPRITE4        ; copia il byte in VSTART4
        MOVE.b        d0,MIOSPRITE5        ; copia il byte in VSTART5
        ADD.B        #15,D0                ; Aggiungi la lunghezza dello sprite per
                           ; determinare la posizione finale (VSTOP)
        move.b        d0,MIOSPRITE4+2        ; Muovi il valore giusto in VSTOP4
        move.b        d0,MIOSPRITE5+2        ; Muovi il valore giusto in VSTOP5
        rts

TABYPOINT2:
        dc.l        TABY+90                ; NOTA: i valori della tabella sono bytes


; Per lo sprite3 ATTACCHED: (ossia Sprite5+Sprite6)

MuoviSpriteY3:
        ADDQ.L        #1,TABYPOINT3         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT3(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTY3        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT3 ; Riparti a puntare dal primo byte (-1)
```

```
NOBSTARTY3:
        moveq        #0,d0               ; Pulisci d0
        MOVE.b        (A0),d0               ; copia il byte dalla tabella in d0
        MOVE.b        d0,MIOSPRITE6        ; copia il byte in VSTART6
        MOVE.b        d0,MIOSPRITE7        ; copia il byte in VSTART7
        ADD.B        #15,D0               ; Aggiungi la lunghezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,MIOSPRITE6+2        ; Muovi il valore giusto in VSTOP6
        move.b        d0,MIOSPRITE7+2        ; Muovi il valore giusto in VSTOP7
        rts

TABYPOINT3:
        dc.l        TABY+140        ; NOTA: i valori della tabella sono bytes



; Tabella con coordinate X dello sprite precalcolate.

TABX:
        incbin        "XCOORDINAT.TAB"        ; 334 valori
FINETABX:


; Tabella con coordinate Y dello sprite precalcolate.

TABY:
        incbin        "YCOORDINAT.TAB"        ; 200 valori
FINETABY:


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

;        Palette della PIC

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                        ; in questo caso e' vuoto,
                                        ; per cui non compare.

;        Palette degli SPRITE attacched
```

```
        dc.w        $1A2,$FFC        ; color17, COLORE 1 per gli sprite attaccati
        dc.w        $1A4,$EEB        ; color18, COLORE 2 per gli sprite attaccati
        dc.w        $1A6,$CD9        ; color19, COLORE 3 per gli sprite attaccati
        dc.w        $1A8,$AC8        ; color20, COLORE 4 per gli sprite attaccati
        dc.w        $1AA,$8B6        ; color21, COLORE 5 per gli sprite attaccati
        dc.w        $1AC,$6A5        ; color22, COLORE 6 per gli sprite attaccati
        dc.w        $1AE,$494        ; color23, COLORE 7 per gli sprite attaccati
        dc.w        $1B0,$384        ; color24, COLORE 7 per gli sprite attaccati
        dc.w        $1B2,$274        ; color25, COLORE 9 per gli sprite attaccati
        dc.w        $1B4,$164        ; color26, COLORE 10 per gli sprite attaccati
        dc.w        $1B6,$154        ; color27, COLORE 11 per gli sprite attaccati
        dc.w        $1B8,$044        ; color28, COLORE 12 per gli sprite attaccati
        dc.w        $1BA,$033        ; color29, COLORE 13 per gli sprite attaccati
        dc.w        $1BC,$012        ; color30, COLORE 14 per gli sprite attaccati
        dc.w        $1BE,$001        ; color31, COLORE 15 per gli sprite attaccati

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! **********

MIOSPRITE0:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.PARI"

MIOSPRITE1:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.DISPARI"

MIOSPRITE2:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.PARI"

MIOSPRITE3:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.DISPARI"

MIOSPRITE4:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.PARI"

MIOSPRITE5:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.DISPARI"

MIOSPRITE6:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.PARI"

MIOSPRITE7:                                ; lunghezza 15 linee
        incbin        "Sprite16Col.DISPARI"


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo listato vengono visualizzati tutti i 4 sprite ATTACCHED a 16 colori.
Gli sprite sono stati salvati (comprese le word di controllo) in file,
usando il comando "WB". Questo per risparmiare spazio nel listato e per
riutilizzare lo sprite attacched in altri listati e varie volte nello stesso
listato, infatti lo stesso sprite (diviso in SPRITE PARI e DISPARI) e'
utilizzato per tutti i quattro sprite.

Per quanto riguarda il movimento degli sprite, ognuno ha una routine di
movimento autonoma, con un puntatore alle tabelle di X e di Y autonomo.
In questo modo, facendo partire il movimento da fasi diverse (ossia punti
diversi della tabella) in ogni sprite, si generano i movimenti piu' disparati.
Le due tabelle X ed Y pero' sono le stesse per tutte le routine, tra una
routine e l'altra cambia solo la posizione di inizio del puntatore, per cui
mentre uno sprite parte dalla posizione X,Y, un'altro parte dalla posizione
X+n, Y+n, creando sprite piu' avanti e piu' indietro nella curva (NEL CASO
DELLA "FILA INDIANA"), oppure traiettorie apparentemente casuali.
E' degna di nota una particolarita' della struttura delle routine in questo
listato: dovendo aspettare la pressione del tasto sinistro e destro piu' volte
per cambiare il movimento degli sprite prima di uscire, sarebbe stato
necessario riscrivere ogni volta i due loop che aspettano la linea $FF del
pennello elettronico e tutti gli 8 "BSR muovisprite":

```
; aspetta linea $FF
; bsr muovisprite
; aspetta il mouse sinistro

; cambia la traiettoria degli sprite

; aspetta linea $FF
; bsr muovisprite
; aspetta il mouse destro

; cambia la traiettoria degli sprite

; aspetta linea $FF
; bsr muovisprite
; aspetta il mouse sinistro

; cambia la traiettoria degli sprite

; aspetta linea $FF
; bsr muovisprite
; aspetta il mouse destro
```

Per risparmiare linee di listato una soluzione e' quella di includere il
loop che aspetta il pennello elettronico per la temporizzazione nella
subroutine BSR muovisprite:

```
; Questa routine esegue le singole routines di movimento degli sprite
; ed include anche il loop di attesa del fotogramma per la temporizzazione.

MuoviGliSprite:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        MuoviGliSprite

        bsr.s        MuoviSpriteX0        ; Muovi lo sprite 0 orizzontalmente
        bsr.w        MuoviSpriteX1        ; Muovi lo sprite 1 orizzontalmente
        bsr.w        MuoviSpriteX2        ; Muovi lo sprite 2 orizzontalmente
        bsr.w        MuoviSpriteX3        ; Muovi lo sprite 3 orizzontalmente
        bsr.w        MuoviSpriteY0        ; Muovi lo sprite 0 verticalmente
        bsr.w        MuoviSpriteY1        ; Muovi lo sprite 1 verticalmente
        bsr.w        MuoviSpriteY2        ; Muovi lo sprite 2 verticalmente
        bsr.w        MuoviSpriteY3        ; Muovi lo sprite 3 verticalmente

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        rts                ; Torna al loop MOUSE
```

```
In questo modo basta aspettare la pressione del tasto del mouse, se non e'
premuto eseguire MuovigliSprite:


Mouse1:
        bsr.w        MuoviGliSprite        ; Attende un fotogramma, muove gli sprite e
                                           ; ritorna.

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse1

        MOVE.L        #TABX+170,TABXPOINT0        ; cambia la traiettoria degli sprite
        ...

Mouse2:
        bsr.w        MuoviGliSprite        ; Attende un fotogramma, muove gli sprite e
                                           ; ritorna.

        btst        #2,$dff016        ; tasto destro del mouse premuto?
        bne.s        mouse2
```

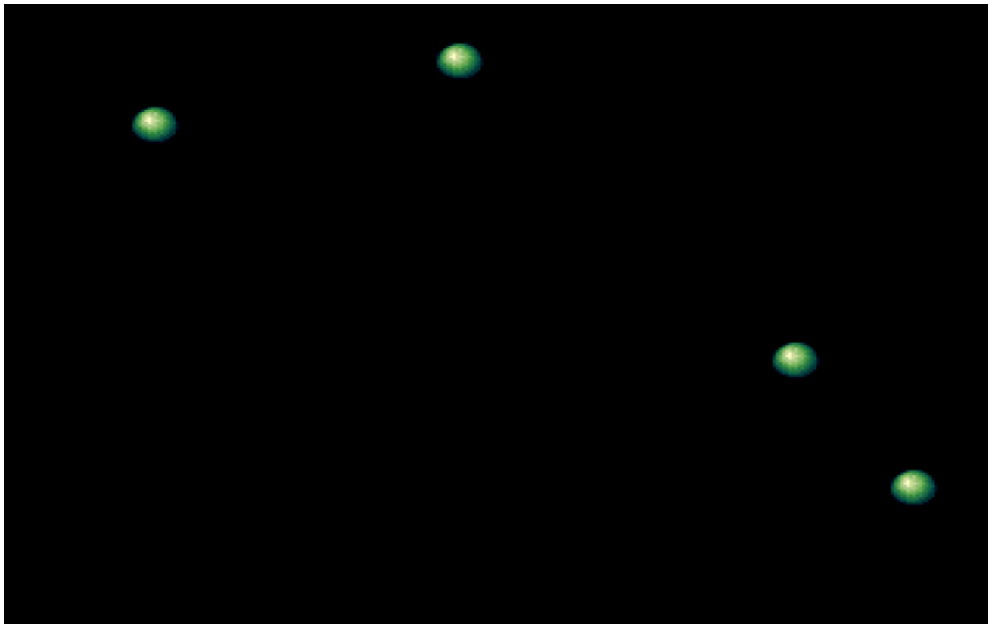


Figura 23.3: Lezione 7h

23.9 Lezione7i

```
; Lezione7i.s        SCORRIMENTO ORIZZONTALE DI UNO SPRITE A PASSI DI 1 PIXEL ALLA
;                VOLTA ANZICHE' DI 2 PIXEL ALLA VOLTA. IN QUESTO MODO LO SCROLL
;                NON VA PIU' A SCATTI.
```

```
        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)         ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0         ; dove puntare
        LEA         BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080         ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006         ; Linea 255?
        bne.s         mouse

        bsr.s         MuoviSpriteX         ; Muovi lo sprite 0 orizzontalmente (FLUIDO)
        bsr.w         MuoviSpriteY         ; Muovi lo sprite 0 verticalmente

Aspetta:
        cmpi.b        #$ff,$dff006         ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001         ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080         ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)         ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)         ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0
```

```
OldCop:
        dc.l        0

; Questa routine sposta  lo sprite agendo sul suo byte HSTART, e sul bit 0 del
; quarto byte di controllo, ossia sul bit basso del valore di HSTART. In questo
; modo lo scorrimento orizzontale procede a scatti di 1 pixel anziche' 2, per
; cui la fluidita' del movimento orizzontale e' raddoppiata e la scattosita'
; vista negli esempi precedenti scompare. La parte di routine che "trasforma"
; il valore della cordinata reale in byte hstart+bit basso puo' essere usata
; in tutti i listati che muovono sprite, tra l'altro con un add.w #128,d0 c'e'
; gia' l'aggiunta dell'offset dall'inizio del video, per cui il valore da dare
; alla routine in entrata e' la coordinata X reale dello schermo LOWRES, per
; cui se mettiamo 0 lo sprite si posiziona al lato sinistro dello schermo, se
; mettiamo 160 si posiziona al centro, con 320 si posiziona all'ultimo pixel
; a destra dello schermo.

MuoviSpriteX:
        ADDQ.L        #2,TABXPOINT         ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq        #0,d0               ; azzeriamo d0
        MOVE.w        (A0),d0               ; poniamo il valore della tabella in d0
        add.w        #128,D0               ; 128 - per centrare lo sprite.
        btst        #0,D0               ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,MIOSPRITE+3        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,MIOSPRITE+3        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w        #1,D0               ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                              ; il valore di HSTART, per "trasformarlo" nel
                              ; valore fa porre nel byte HSTART, senza cioe'
                              ; il bit basso.
        move.b        D0,HSTART        ; Poniamo il valore XX nel byte HSTART
        rts

TABXPOINT:
        dc.l        TABX-2               ; NOTA: i valori della tabella qua sono word,

; Tabella con coordinate X dello sprite precalcolate. Questa tabella contiene
; i valori REALI delle coordinate dello schermo, non i valori "dimezzati" per
; lo scorrimento a scatti di due pixel come abbiamo visto fino ad ora.
; Essendo i valori possibili piu' di 256, si supera la grandezza di un byte,
; per cui la tabella e' composta di WORD, le quali possono contenere tali
; valori. E' la routine "MuoviSpriteX" che si occupa di prelevare la WORD dalla
; tabella, e di dividere il numero in BIT BASSO per lo scorrimento di 1 pixel
; anziche' 2, e negli altri 8 bit, che si occupano degli "scatti di 2 pixel",
; cioe' il byte HSTART che abbiamo usato da solo fino ad ora.
; Da notare che la posizione X per far entrare lo sprite nella finestra video
; deve essere compresa tra 0 e 320, cioe' la posizione effettiva all'interno
; dello schermo, l'offset dall'inizio del video di 128 (cioe' $40*2) viene
; aggiunto dalla routine.
; Bisogna ricordarsi anche che lo sprite e' largo 16 pixel e che la
; coordinata X si riferisce al suo angolo sinistro, per cui se diamo coordinate
; superiori a 320-16, ossia superiori a 304, lo sprite risultera' parzialmente
```

```
; fuori dallo schermo.
; Nella tabella infatti ci sono byte non piu' grandi di 304 e non piu'
; piccoli di zero.


; Ecco come ottenere la tabella:

;                                    ___304
; DEST> tabx                          /   \ 152 (304/2)
; BEG> 0                       \___/     0
; END> 360
; AMOUNT> 150
; AMPLITUDE> 304/2 ; ampiezza sia sopra zero che sotto zero, allora
                        ; bisogna che faccia meta' sopra zero e meta' sotto,
                        ; ossia dividiamo per 2 l'AMPIEZZA
; YOFFSET> 304/2        ; e spostiamo tutto sopra per trasformare -152 in 0
; SIZE (B/W/L)> w
; MULTIPLIER> 1

TABX:
        incbin        "xcoordinatok.tab"        ; 150 valori .W
FINETABX:

; Questa routine sposta in alto e in basso lo sprite agendo sui suoi byte
; VSTART e VSTOP, ossia i byte della sua posizione Y di inizio e fine,
; immettendoci delle coordinate gia' stabilite nella tabella TABY

MuoviSpriteY:
        ADDQ.L        #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte (-1)
NOBSTARTY:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte dalla tabella in d0
        MOVE.b        d0,VSTART        ; copia il byte in VSTART
        ADD.B        #13,D0                ; Aggiungi la lunghezza dello sprite per
                              ; determinare la posizione finale (VSTOP)
        move.b        d0,VSTOP        ; Muovi il valore giusto in VSTOP
        rts

TABYPOINT:
        dc.l        TABY-1                ; NOTA: i valori della tabella qua sono bytes,
                              ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                              ; e non #2 come per quando sono word o con #4
                              ; come quando sono longword.

; Tabella con coordinate Y dello sprite precalcolate.
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $2c e $f2, infatti nella tabella ci sono byte non
; piu' grandi di $f2 e non piu' piccoli di $2c.

TABY:
        dc.b        $8E,$91,$94,$97,$9A,$9D,$A0,$A3,$A6,$A9,$AC,$AF ; ondeggio
        dc.b        $B2,$B4,$B7,$BA,$BD,$BF,$C2,$C5,$C7,$CA,$CC,$CE ; 200 valori
        dc.b        $D1,$D3,$D5,$D7,$D9,$DB,$DD,$DF,$E0,$E2,$E3,$E5
        dc.b        $E6,$E7,$E9,$EA,$EB,$EC,$EC,$ED,$EE,$EE,$EF,$EF
        dc.b        $EF,$EF,$F0,$EF,$EF,$EF,$EF,$EE,$EE,$ED,$EC,$EC
        dc.b        $EB,$EA,$E9,$E7,$E6,$E5,$E3,$E2,$E0,$DF,$DD,$DB
        dc.b        $D9,$D7,$D5,$D3,$D1,$CE,$CC,$CA,$C7,$C5,$C2,$BF
```

```
        dc.b        $BD,$BA,$B7,$B4,$B2,$AF,$AC,$A9,$A6,$A3,$A0,$9D
        dc.b        $9A,$97,$94,$91,$8E,$8B,$88,$85,$82,$7F,$7C,$79
        dc.b        $76,$73,$70,$6D,$6A,$68,$65,$62,$5F,$5D,$5A,$57
        dc.b        $55,$52,$50,$4E,$4B,$49,$47,$45,$43,$41,$3F,$3D
        dc.b        $3C,$3A,$39,$37,$36,$35,$33,$32,$31,$30,$30,$2F
        dc.b        $2E,$2E,$2D,$2D,$2D,$2D,$2C,$2D,$2D,$2D,$2D,$2E
        dc.b        $2E,$2F,$30,$30,$31,$32,$33,$35,$36,$37,$39,$3A
        dc.b        $3C,$3D,$3F,$41,$43,$45,$47,$49,$4B,$4E,$50,$52
        dc.b        $55,$57,$5A,$5D,$5F,$62,$65,$68,$6A,$6D,$70,$73
        dc.b        $76,$79,$7C,$7F,$82,$85,$88,$8b
FINETABY:


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
```

```
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0         ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

23.10 Lezione7l

```
; Lezione7l.s        SCORRIMENTO VERTICALE DI UNO SPRITE SOTTO LA LINEA $FF


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1         ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
```

```
        cmpi.b      #$aa,$dff006        ; Linea $aa?
        bne.s       mouse

        btst        #2,$dff016
        beq.s       aspetta
        bsr.w       MuoviSpriteY        ; Muovi lo sprite 0 verticalmente (oltre $FF)

Aspetta:
        cmpi.b      #$aa,$dff006        ; linea $aa?
        beq.s       Aspetta

        btst        #6,$bfe001          ; mouse premuto?
        bne.s       mouse

        move.l      OldCop(PC),$dff080  ; Puntiamo la cop di sistema
        move.w      d0,$dff088          ; facciamo partire la vecchia cop

        move.l      4.w,a6
        jsr         -$7e(a6)            ; Enable
        move.l      gfxbase(PC),a1
        jsr         -$19e(a6)           ; Closelibrary
        rts

;       Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine sposta in alto e in basso lo sprite agendo sui suoi byte
; VSTART e VSTOP, ossia i byte della sua posizione Y di inizio e fine, nonche'
; i bit alti delle coordinate VSTART/VSTOP, permettendo il posizionamento
; dello sprite anche nelle linee sotto $FF. La coordinata iniziale deve essere
; in formato WORD, da 0 a $FF per rimanere nello schermo normale (l'offset di
; $2c viene aggiunto dalla routine) oppure si puo' andare oltre $FF per
; far andare lo sprite fino al limite hardware negli schermi overscan.

MuoviSpriteY:
        ADDQ.L      #2,TABYPOINT        ; Fai puntare alla word successiva
        MOVE.L      TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L       #FINETABY-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S       NOBSTARTY       ; non ancora? allora continua
        MOVE.L      #TABY-2,TABYPOINT ; Riparti a puntare dalla prima word (-2)
NOBSTARTY:
        moveq       #0,d0               ; Pulisci d0
        MOVE.w      (A0),d0               ; copia la word dalla tabella in d0
        ADD.W       #$2c,d0               ; aggiungi l'offset dell'inizio dello schermo
        MOVE.b      d0,VSTART       ; copia il byte (bits 0-7) in VSTART
        btst.l      #8,d0               ; la posizione e' maggiore di 255? ($FF)
        beq.s       NonVSTARTSET
        bset.b      #2,MIOSPRITE+3      ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s       ToVSTOP
NonVSTARTSET:
        bclr.b      #2,MIOSPRITE+3      ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w       #13,D0              ; Aggiungi la lunghezza dello sprite per
```

```
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,VSTOP        ; Muovi il valore giusto (bits 0-7) in VSTOP
        btst.l        #8,d0                ; la posizione e' maggiore di 255? ($FF)
        beq.s        NonVSTOPSET
        bset.b        #1,MIOSPRITE+3        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w        VstopFIN
NonVSTOPSET:
        bclr.b        #1,MIOSPRITE+3        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:
        rts

TABYPOINT:
        dc.l        TABY-2                ; NOTA: i valori della tabella qua sono bytes,
                                ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                                ; e non #2 come per quando sono word o con #4
                                ; come quando sono longword.

; Tabella con coordinate Y dello sprite precalcolate.
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $0 e $ff, infatti l'offset di $2c viene aggiunto
; dalla routine. Se non si usano schermi overscan, ossia non piu' lunghi di
; 255 linee, si puo' usare una tabella di valori dc.b (da $00 a $FF)


; Come rifarsi la tabella:

; BEG> 0
; END> 360
; AMOUNT> 200
; AMPLITUDE> $f0/2
; YOFFSET> $f0/2
; SIZE (B/W/L)> b
; MULTIPLIER> 1


TABY:
        DC.W        $7A,$7E,$81,$85,$89,$8D,$90,$94,$98,$9B,$9F,$A2,$A6,$A9,$AD
        DC.W        $B0,$B3,$B7,$BA,$BD,$C0,$C3,$C6,$C9,$CC,$CE,$D1,$D3,$D6,$D8
        DC.W        $DA,$DC,$DE,$E0,$E2,$E4,$E5,$E7,$E8,$EA,$EB,$EC,$ED,$EE,$EE
        DC.W        $EF,$EF,$F0,$F0,$F0,$F0,$F0,$F0,$EF,$EF,$EE,$EE,$ED,$EC,$EB
        DC.W        $EA,$E8,$E7,$E5,$E4,$E2,$E0,$DE,$DC,$DA,$D8,$D6,$D3,$D1,$CE
        DC.W        $CC,$C9,$C6,$C3,$C0,$BD,$BA,$B7,$B3,$B0,$AD,$A9,$A6,$A2,$9F
        DC.W        $9B,$98,$94,$90,$8D,$89,$85,$81,$7E,$7A,$76,$72,$6F,$6B,$67
        DC.W        $63,$60,$5C,$58,$55,$51,$4E,$4A,$47,$43,$40,$3D,$39,$36,$33
        DC.W        $30,$2D,$2A,$27,$24,$22,$1F,$1D,$1A,$18,$16,$14,$12,$10,$0E
        DC.W        $0C,$0B,$09,$08,$06,$05,$04,$03,$02,$02,$01,$01,$00,$00,$00
        DC.W        $00,$00,$00,$01,$01,$02,$02,$03,$04,$05,$06,$08,$09,$0B,$0C
        DC.W        $0E,$10,$12,$14,$16,$18,$1A,$1D,$1F,$22,$24,$27,$2A,$2D,$30
        DC.W        $33,$36,$39,$3D,$40,$43,$47,$4A,$4E,$51,$55,$58,$5C,$60,$63
        DC.W        $67,$6B,$6F,$72,$76
FINETABY:

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
```

```
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

Questo esempio e‘ quasi identico a quello del sorgente LEZIONE7d. In questo

esempio pero‘ la posizione verticale dello sprite puo‘ andare oltre la linea
255. Vi ricordo che poiche‘ la finestra video inizia alle coordinate ($40,$2c)
la linea 255 corrisponde alla 211-esima linea visibile sullo schermo
(infatti 255-$2c=211). Quindi se vogliamo che il nostro sprite possa muoversi
per tutte le 256 linee visibili sullo schermo, e‘ necessario che la posizione
verticale raggiunga il valore 299=$12b. Questo valore e‘ troppo grande per
essere contenuto in un byte, sono necessari 9 bit. Per specificare la posizione
Y di inizio dello sprite si usa quindi, oltre agli 8 bit del byte VSTART
(che abbiamo usato finora), un ulteriore bit, precisamente il bit 2 del byte
VHBITS, ovvero il quarto byte di controllo. Lo stesso discorso vale per la
posizione di fine dello sprite, solo che si usa il bit 1 del byte VHBITS.
Nella tabella invece le posizioni verticali sono memorizzate come word.
La routine che legge le coordinate verticali dalla tabella controlla se
i valori letti sono maggiori di 255; se questo accade mette a 1 il bit giusto
del registro VHBITS, altrimenti lo azzera. Da notare che il controllo viene
fatto indipendentemente per la posizione di inizio e per quella di fine;
puo‘ accadere infatti che uno sprite inizi ad una posizione minore di 255,
pero‘ finisca ad una posizione maggiore di 255. In questo caso il bit 2 di
VHBITS viene azzerato, mentre il bit 1 di VHBITS viene settato a 1.

23.11 Lezione7m

```
; Lezione7m.s        Posizionamento degli sprite tramite una routine universale
; Questo esempio mostra una routine universale per spostare gli sprite che
; considera tutti i bit delle posizioni orizzontali e verticali degli sprite.
; Inoltre aggiunge automaticamente gli offsets (128 per le coordinate
; orizzontali, $2c per le verticali).
; In questo modo le coordinate nelle tabelle possono essere quelle reali,
; cioe‘ da 0 a 320 per le coordinate orizzontali e da 0 a 256 per le
; coordinate verticali


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
```

```
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$aa,$dff006        ; Linea $aa?
        bne.s         mouse

        btst         #2,$dff016
        beq.s         aspetta
        bsr.w         MuoviSprite         ; Muovi lo sprite 0

Aspetta:
        cmpi.b        #$aa,$dff006        ; linea $aa?
        beq.s         Aspetta

        btst         #6,$bfe001         ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)         ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)         ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0

OldCop:
        dc.l         0


;        Per muovere correttamente lo sprite, prima leggiamo le tabelle per
;        sapere quali posizioni deve assumere lo sprite, poi comunichiamo tali
;        posizioni, nonche' l'indirizzo e l'altezza degli sprite, alla routine
;        UniMuoviSprite, tramite i registri a1,d0,d1,d2

MuoviSprite:
        bsr.s         LeggiTabelle         ; Legge le posizioni X ed Y dalle tabelle,
                                  ; mettendo nel registro a1 l'indirizzo dello
                                  ; sprite, in d0 la pos. Y, in d1 la pos. X
                                  ; e in d2 l'altezza dello sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

        bsr.w         UniMuoviSprite  ; esegue la routine universale che posiziona
                                  ; lo sprite
```

```
        rts



; Questa routine legge dalle 2 tabelle le coordinate reali degli sprite.
; Cioe' la coordinata X varia da 0 a 320 e la Y da 0 a 256 (senza overscan).
; Poiche' in questo esempio non usiamo l'overscan, la tabella delle coordinate
; Y e' una tabella di byte. La tabella delle coordinate X, invece e' fatta
; di word perche' deve contenere anche valori maggiori di 256.
; Questa routine, pero' non posiziona direttamente lo sprite. Essa si limita
; semplicemente a farlo fare alla routine universale, comunicandogli le
; coordinate tramite i registri d0 e d1

LeggiTabelle:
        ADDQ.L        #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte
NOBSTARTY:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte della tabella, cioe' la
                                ; coordinata Y in d0 in modo da farla
                                ; trovare alla routine universale

        ADDQ.L        #2,TABXPOINT         ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq        #0,d1                ; azzeriamo d1
        MOVE.w        (A0),d1                ; poniamo il valore della tabella, cioe'
                                ; la coordinata X in d1

        lea        MIOSPRITE,a1        ; indirizzo dello sprite in A1
        moveq        #13,d2                ; altezza dello sprite in d2
        rts


TABYPOINT:
        dc.l        TABY-1                ; NOTA: i valori della tabella qua sono bytes,
                                ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                                ; e non #2 come per quando sono word o con #4
                                ; come quando sono longword.
TABXPOINT:
        dc.l        TABX-2                ; NOTA: i valori della tabella qua sono word,

; Tabella con coordinate Y dello sprite precalcolate.
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $0 e $ff, infatti l'offset di $2c viene aggiunto
; dalla routine. Se non si usano schermi overscan, ossia non piu' lunghi di
; 255 linee, si puo' usare una tabella di valori dc.b (da $00 a $FF)


; Come rifarsi la tabella:

; BEG> 0
; END> 360
```

```
; AMOUNT> 200
; AMPLITUDE> $f0/2
; YOFFSET> $f0/2
; SIZE (B/W/L)> b
; MULTIPLIER> 1


TABY:
        incbin        "ycoordinatok.tab"        ; 200 valori .B
FINETABY:

; Tabella con coordinate X dello sprite precalcolate. Questa tabella contiene
; i valori REALI delle coordinate dello schermo, non i valori "dimezzati" per
; lo scorrimento a scatti di due pixel come abbiamo visto fino ad ora.
; Nella tabella infatti ci sono byte non piu' grandi di 304 e non piu'
; piccoli di zero.

TABX:
        incbin        "xcoordinatok.tab"        ; 150 valori .W
FINETABX:



; Routine universale di posizionamento degli sprite.
; Questa routine modifica la posizione dello sprite il cui indirizzo e'
; contenuto nel registro a1 e la cui altezza e' contenuta nel registro d2,
; e posiziona lo sprite alle coordinate Y e X contenute rispettivamente nei
; registri d0 e d1.
; Prima di chiamare questa routine e' necessario mettere l'indirizzo dello
; sprite nel registro a1, la sua altezza nel registro d2, la coordinata Y nel
; registro d0, la X nel registro d1

; Questa procedura e' chiamata "passaggio di parametri".
; Notate che questa routine modifica i registri d0 e d1.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W        #$2c,d0                ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite

        MOVE.b        d0,(a1)                ; copia il byte in VSTART
        btst.l        #8,d0
        beq.s        NonVSTARTSET
        bset.b        #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s        ToVSTOP
NonVSTARTSET:
        bclr.b        #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w        D2,D0                ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,2(a1)        ; Muovi il valore giusto in VSTOP
```

```
        btst.l        #8,d0
        beq.s         NonVSTOPSET
        bset.b        #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w         VstopFIN
NonVSTOPSET:
        bclr.b        #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale

        add.w         #128,D1               ; 128 - per centrare lo sprite.
        btst         #0,D1                ; bit basso della coordinata X azzerato?
        beq.s         BitBassoZERO
        bset         #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s         PlaceCoords

BitBassoZERO:
        bclr         #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w         #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                  ; il valore di HSTART, per "trasformarlo" nel
                                  ; valore fa porre nel byte HSTART, senza cioe'
                                  ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w         $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w         $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w         $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w         $13e,0

        dc.w         $8E,$2c81        ; DiwStrt
        dc.w         $90,$2cc1        ; DiwStop
        dc.w         $92,$38               ; DdfStart
        dc.w         $94,$d0               ; DdfStop
        dc.w         $102,0               ; BplCon1
        dc.w         $104,0               ; BplCon2
        dc.w         $108,0               ; Bpl1Mod
        dc.w         $10a,0               ; Bpl2Mod

                     ; 5432109876543210
        dc.w         $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w         $180,$000        ; color0        ; sfondo nero
        dc.w         $182,$123        ; color1        ; colore 1 del bitplane, che
                                               ; in questo caso e' vuoto,
                                               ; per cui non compare.

        dc.w         $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w         $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w         $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w         $FFFF,$FFFE        ; Fine della copperlist
```

```
; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questa lezione presentiamo una routine universale per spostare gli sprite,
chiamata "UniMuoviSprite".
Questa routine si occupa di tutti gli aspetti del posizionamento degli sprite,
gestisce correttamente tutti i bit della posizione e aggiunge anche gli offset
in modo tale che nelle tabelle possono essere memorizzate le coordinate reali
degli sprite.
Questa routine funziona con un qualsiasi sprite. Infatti l'indirizzo dello
sprite non e' fisso, ma viene letto nel registro a1.
Questo vuol dire che:

  VSTART si trova all'indirizzo contenuto in a1

  HSTART si trova nel byte seguente, cioe' nell'indirizzo contenuto in a1 +1

  VSTOP si trova 2 byte dopo, cioe' nell'indirizzo contenuto in a1 +2

 il quarto byte si trova 3 bytes dopo, cioe' nell'indirizzo contenuto in a1 +3.

UniMuoviSprite accede a questi byte mediante l'indirizzamento indiretto
a registro con spiazzamento:

 per accedere a VSTART si usa (a1)
 per accedere a HSTART si usa 1(a1)
 per accedere a VSTOP si usa 2(a1)
 per accedere al quarto byte di controllo si usa 3(a1)

Anche l'altezza dello sprite non e' fissa, ma e' contenuta nel registro d2.
In questo modo la routine puo' essere usata per muovere sprites di diversa
altezza. Inoltre questa routine non legge direttamente le coordinate

```
dalla tabella, ma li prende dai registri d0 e d1.

Chi mette i dati in questi registri? Se ne occupa un'altra routine
"LeggiTabelle" che preleva le coordinate dalle tabelle, le mette nei registri
d0 e d1, e esegue la routine "UniMuoviSprite". In pratica abbiamo diviso
i compiti tra le 2 routine, come se fossero 2 impiegati. La routine
"Leggitabelle" svolge il suo compito, poi dice:"Ehi routine UniMUoviSprite,
eccoti lo sprite da mouvere, ti spedisco l'indirizzo nel registro a1.
Ti spedisco in d2 l'altezza dello sprite.
Eccoti inoltre le coordinate, te le spedisco attraverso i registri d0 e d1.
Sai tu cosa farne!".
La routine "UniMuoviSprite" riceve l'indirizzo dello sprite e le coordinate e
le mette nei giusti byte dello sprite.
La "spedizione" delle coordinate attraverso i registri si chiama "passaggio di
parametri".
La divisione dei compiti e' una cosa molto comoda. Supponiamo infatti di voler
muovere uno sprite utilizzando una tabella per le Y, mentre per le X una
routine di incremento e decremento continuo ADDQ/AUBQ separata, in modo da
realizzare uno sprite che si muove sempre da sinistra a destra, ma che oscilla
in alto e in basso.
Poiche' la routine universale che abbimo appena visto prende le coordinate dai
registri, senza interessarsi se prima questi dati provenivano da una tabella,
possiamo usarla di nuovo cosi' com'e' in questo listato, senza doverla
modificare per niente.
Inoltre, poiche' prende l'indirizzo dello sprite da un registro, e la sua
altezza da un altro, puo' essere usata per qualsiasi sprite.
D'ora in poi, per ogni altro esempio sugli sprite, useremo quindi sempre la
routine "UniMuoviSprite", senza doverla modificare ogni volta.
```

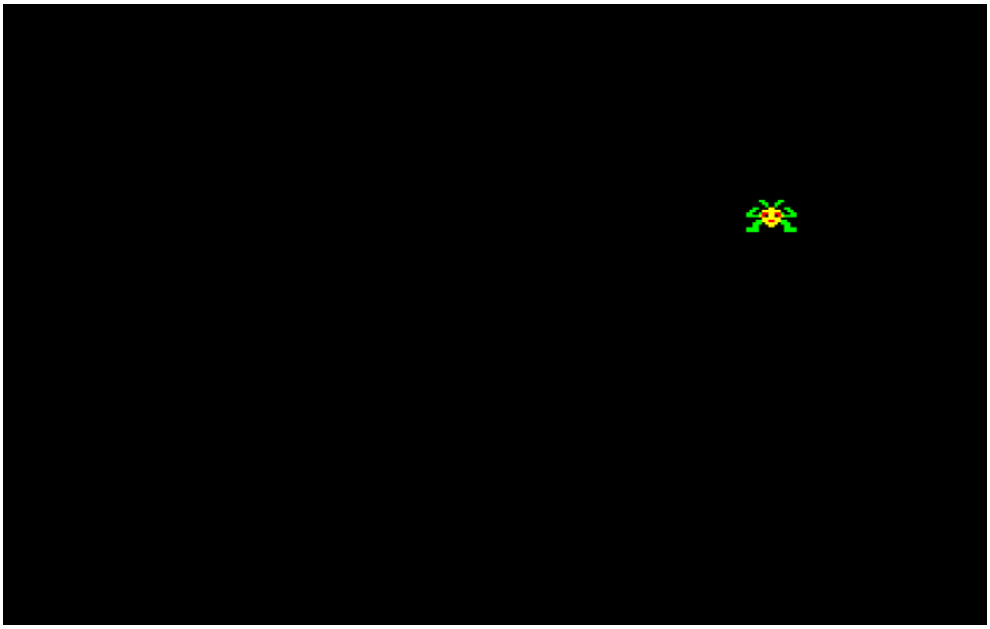


Figura 23.4: Lezione 7m

23.12 Lezione7n

```
; Lezione7n.s - esempio di applicazione della routine universale:
;               uno sprite che rimbalza


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$aa,$dff006        ; Linea $aa?
        bne.s        mouse

        btst        #2,$dff016
        beq.s        aspetta
        bsr.w        MuoviSprite        ; Muovi lo sprite 0

Aspetta:
        cmpi.b        #$aa,$dff006        ; linea $aa?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts
```

```
;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine cambia le coordinate dello sprite aggiungendo una velocita‘
; costante sia in verticale che in orizzontale. Inoltre quando lo sprite tocca
; uno dei bordi, la routine provvede a invertire la direzione.
; Per comprendere questa routine occorre sapere che l’istruzione "NEG" serve
; a trasformare un numero positivo in negativo e viceversa.

MuoviSprite:
        move.w        sprite_y(PC),d0        ; leggi la vecchia posizione
        add.w        speed_y(PC),d0        ; aggiungi la velocita‘
        btst        #15,d0                ; se il bit 15 e’ settato, il numero e’
                            ; diventato negativo. E’ diventato negativo?
        beq.s        no_tocca_sopra        ; se >0 va bene
        neg.w        speed_y                ; se <0 abbiamo toccato il bordo superiore
                            ; allora inverti la direzione
        bra.s        Muovisprite        ; ricalcola la nuova posizione

no_tocca_sopra:
        cmp.w        #243,d0        ; quando la posizione vale 256-13=243, lo sprite
                    ; tocca il bordo inferiore
        blo.s        no_tocca_sotto
        neg.w        speed_y                ; se lo sprite tocca il bordo inferiore,
                            ; inverti la velocita‘
        bra.s        Muovisprite        ; ricacola la nuova posizione

no_tocca_sotto:
        move        d0,sprite_y        ; aggiorna la posizione
posiz_x:
        move.w        sprite_x(PC),d1        ; leggi la vecchia posizione
        add.w        speed_x(PC),d1        ; aggiungi la velocita‘
        btst        #15,d0                ; se il bit 15 e’ settato, il numero e’
                            ; diventato negativo. E’ diventato negativo?
        beq.s        no_tocca_sinistra
        neg.w        speed_x                ; se <0 tocca a sinistra: inverti la direzione
        bra.s        posiz_x                ; ricalcola nuova posizione oriz.

no_tocca_sinistra:
        cmp.w        #304,d1        ; quando la posizione vale 320-16=304, lo sprite
                    ; tocca il bordo destro
        blo.s        no_tocca_destra
        neg.w        speed_x                ; se tocca a destra, inverti la direzione
        bra.s        posiz_x                ; ricalcola nuova posizione oriz.

no_tocca_destra:
        move.w        d1,sprite_x        ; aggiorna la posizione

        lea        miosprite,a1        ; indirizzo sprite
        moveq        #13,d2                ; altezza sprite
        bsr.s        UniMuoviSprite  ; esegue la routine universale che posiziona
                                ; lo sprite
        rts
```

```
SPRITE_Y:
                DC.W        10        ; posizione sprite
SPRITE_X:
                DC.W        0
SPEED_Y:
                dc.w        -4                ; velocita‘ sprite
SPEED_X:
                dc.w        3

; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W        #$2c,d0                ; aggiungi l’offset dell’inizio dello schermo

; a1 contiene l’indirizzo dello sprite
        MOVE.b        d0,(a1)                ; copia il byte in VSTART
        btst.l        #8,d0
        beq.s        NonVSTARTSET
        bset.b        #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s        ToVSTOP
NonVSTARTSET:
        bclr.b        #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w        D2,D0                ; Aggiungi l’altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l        #8,d0
        beq.s        NonVSTOPSET
        bset.b        #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w        VstopFIN
NonVSTOPSET:
        bclr.b        #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w        #128,D1                ; 128 - per centrare lo sprite.
        btst        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe’
                                ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts
```

```
        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
```

```
                                    ; e' necessario che ci siano bitplanes
                                    ; abilitati
BITPLANE:
        ds.b        40*256              ; bitplane azzerato lowres

        end
```

In questo esempio mostriamo un modo diverso di muovere gli sprite, senza
usare le tabelle.
In qesto esempio il nostro sprite si muove in modo rettilineo, con velocita'
costante sia per la posizione orizzontale, sia per quella verticale.
La velocita' non e' altro che un numero, contenuto in una locazione di memoria,
che viene aggiunto ogni volta alla posizione che lo sprite occupava in
precedenza, calcolando cosi' la nuova posizione.
Se la velocita' e' un numero positivo, aumentera' ogni volta la posizione
dello sprite, spostandolo verso destra (o in basso nel caso Y).
Se la velocita' e' un numero negativo, diminuira' ogni volta la posizione
dello sprite, spostandolo verso sinistra (o in alto nel caso Y).
Quando lo sprite tocca uno dei bordi e' necessario invertire la direzione
in cui si sta' spostando. Per fare cio' e' sufficente cambiare di segno la
velocita', trasformandola cioe' da positiva in negativa o viceversa.
Di questo si occupa l'istruzione NEG che cambia appunto il segno di un numero
contenuto in un registro o in una locazione di memoria.

23.13 Lezione7o

```
; Lezione7o.s - esempio di applicazione della routine universale:
;               due sprite mossi dalla stessa routine


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6               ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1        ; Nome lib
        jsr         -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA         BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0               ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        addq.l        #8,a1                       ; puntatore a sprite 1
        MOVE.L        #MIOSPRITE2,d0               ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
```

```
        swap        d0
        move.w       d0,2(a1)

        move.l       #COPPERLIST,$dff080        ; nostra COP
        move.w       d0,$dff088                ; START COP
        move.w       #0,$dff1fc                ; NO AGA!
        move.w       #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b       #$aa,$dff006        ; Linea $aa?
        bne.s        mouse

        btst        #2,$dff016
        beq.s        aspetta
        bsr.w        MuoviSprite        ; Muovi lo sprite 0

Aspetta:
        cmpi.b       #$aa,$dff006        ; linea $aa?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


; Questa routine legge dalle 2 tabelle le coordinate reali degli sprite.
; Cioe‘ la coordinata X varia da 0 a 320 e la Y da 0 a 256 (senza overscan).
; Poiche‘ in questo esempio non usiamo l’overscan, la tabella delle coordinate
; Y e‘ una tabella di byte. La tabella delle coordinate X, invece e‘ fatta
; di word perche‘ deve contenere anche valori maggiori di 256.
; Questa routine, pero‘ non posiziona direttamente lo sprite. Essa si limita
; semplicemente a farlo fare alla routine universale, comunicandogli le
; coordinate tramite i registri d0 e d1

MuoviSprite:
        ADDQ.L       #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L       TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all’ultimo byte della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L       #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte
NOBSTARTY:
        moveq        #0,d0                ; Pulisci d0
```

```
        MOVE.b        (A0),d0               ; copia il byte della tabella, cioe' la
                              ; coordinata Y in d0 in modo da farla
                              ; trovare alla routine universale

        ADDQ.L        #2,TABXPOINT         ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                               ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq        #0,d1                ; azzeriamo d1
        MOVE.w        (A0),d1                ; poniamo il valore della tabella, cioe'
                              ; la coordinata X in d1

        lea        MIOSPRITE,a1        ; indirizzo dello sprite in A1
        moveq        #13,d2                ; altezza dello sprite in d2

        bsr.w        UniMuoviSprite  ; esegue la routine universale che posiziona
                              ; lo sprite
; secondo sprite

        ADDQ.L        #1,TABYPOINT2         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT2(PC),A0 ; indirizzo contenuto in long TABXPOINT
                               ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S        NOBSTARTY2        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT2 ; Riparti a puntare dal primo byte
NOBSTARTY2:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte della tabella, cioe' la
                              ; coordinata Y in d0 in modo da farla
                              ; trovare alla routine universale

        ADDQ.L        #2,TABXPOINT2         ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT2(PC),A0 ; indirizzo contenuto in long TABXPOINT
                               ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX2        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT2 ; Riparti a puntare dalla prima word-2
NOBSTARTX2:
        moveq        #0,d1                ; azzeriamo d1
        MOVE.w        (A0),d1                ; poniamo il valore della tabella, cioe'
                              ; la coordinata X in d1

        lea        MIOSPRITE2,a1        ; indirizzo dello sprite in A1
        moveq        #8,d2                ; altezza dello sprite in d2

        bsr.w        UniMuoviSprite  ; esegue la routine universale che posiziona
                              ; lo sprite
        rts

; puntatori alle tabelle del primo sprite

TABYPOINT:
        dc.l        TABY-1
TABXPOINT:
        dc.l        TABX-2

; puntatori alle tabelle del secondo sprite

TABYPOINT2:
```

```
        dc.l        TABY+40-1
TABXPOINT2:
        dc.l        TABX+96-2

; Tabella con coordinate Y dello sprite precalcolate.
TABY:
        incbin       "ycoordinatok.tab"        ; 200 valori .B
FINETABY:

; Tabella con coordinate X dello sprite precalcolate.
TABX:
        incbin       "xcoordinatok.tab"        ; 150 valori .W
FINETABX:

; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W       #$2c,d0               ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b       d0,(a1)               ; copia il byte in VSTART
        btst.l       #8,d0
        beq.s       NonVSTARTSET
        bset.b       #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s       ToVSTOP
NonVSTARTSET:
        bclr.b       #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w       D2,D0                ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b       d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l       #8,d0
        beq.s       NonVSTOPSET
        bset.b       #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w       VstopFIN
NonVSTOPSET:
        bclr.b       #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w       #128,D1               ; 128 - per centrare lo sprite.
        btst        #0,D1               ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
```

```
                            ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.
```

```
MIOSPRITE2:                ; lunghezza 8 linee
VSTART2:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART2:
        dc.b $60+(14*2)        ; Pos. orizzontale (da $40 a $d8)
VSTOP2:
        dc.b $68        ; $60+8=$68        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0011111111111100,%1100001000100011
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0111111111111110,%1000000111000001
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                            ; perche' per vedere gli sprite
                                            ; e' necessario che ci siano bitplanes
                                            ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo esempio mostriamo la versatilita‘ della routine UniMuoviSprite. Abbiamo 2 sprites di forma e altezza diversa, ed entrambi sono mossi sullo schermo dalla routine. Infatti la routine Muovisprite, legge dalle tabelle le coordinate degli sprites, e poi chiama per ognuno degli sprite la routine UniMuoviSprite. Notate come ogni volta la routine MuoviSprite metta nel registro a1 l’indirizzo dello sprite (che e‘ ovviamente diverso per i 2 sprites). Poiche‘ inoltre i 2 sprite hanno altezze diverse, prima di chiamare UnMuoviSprite ogni volta viene messo in d2 un valore diverso, cioe‘ quello dell’altezza di ciascun sprite.

23.14 Lezione7p

```
; Lezione7p.s - esempio di applicazione della routine universale:
;               4 SPRITE A 4 COLORI AFFIANCATI PER FORMARE UNA FIGURA
;                LARGA 64 PIXEL.
;                 USANDO DUE TABELLE DI VALORI (ossia di coordinate verticali
;                e orizzontali) PRESTABILITI.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"
```

```
        MOVE.L      #BITPLANE,d0        ; dove puntare
        LEA       BPLPOINTERS,A1       ; puntatori COP
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)

;       Puntiamo 4 sprite


        MOVE.L      #MIOSPRITE,d0              ; indirizzo dello sprite in d0
        LEA       SpritePointers,a1       ; Puntatori in copperlist
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE1,d0              ; indirizzo dello sprite in d0
        addq.w      #8,a1                      ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE2,d0              ; indirizzo dello sprite in d0
        addq.w      #8,a1                      ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE3,d0              ; indirizzo dello sprite in d0
        addq.w      #8,a1                      ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)

        move.l      #COPPERLIST,$dff080       ; nostra COP
        move.w      d0,$dff088               ; START COP
        move.w      #0,$dff1fc               ; NO AGA!
        move.w      #$c00,$dff106               ; NO AGA!


Mouse1:
        cmpi.b      #$ff,$dff006        ; Linea 255?
        bne.s       Mouse1

        bsr.w       MuoviGliSprite       ; muove tutti gli sprite

Aspetta:
        cmpi.b      #$ff,$dff006        ; linea 255?
        beq.s       Aspetta

        btst       #6,$bfe001       ; tasto sinistro del mouse premuto?
        bne.s       mouse1

        move.l      OldCop(PC),$dff080       ; Puntiamo la cop di sistema
        move.w      d0,$dff088               ; facciamo partire la vecchia cop


        move.l      4.w,a6
        jsr       -$7e(a6)       ; Enable
        move.l      gfxbase(PC),a1
        jsr       -$19e(a6)       ; Closelibrary
        rts

;       Dati
```

```
GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


; Questa routine legge dalla tabella le coordinate dello sprite 0, lo muove
; usando la routine Universale che abbiamo visto in lezione7m, e poi sposta
; anche gli altri sprite. Gli altri sprite avranno la stessa coordinata
; verticale del primo e saranno affiancati distando 16 pixel l'uno dall'altro.
; la posiz. orizzontale dello sprite 1 e' 16 pixel piu' a destra dello sprite 0
; la posiz. orizzontale dello sprite 2 e' 16 pixel piu' a destra dello sprite 1
; la posiz. orizzontale dello sprite 3 e' 16 pixel piu' a destra dello sprite 2

MuoviGliSprite:
        ADDQ.L       #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L       TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L       #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S       NOBSTARTY        ; non ancora? allora continua
        MOVE.L       #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte
NOBSTARTY:
        moveq       #0,d4               ; Pulisci d4
        MOVE.b       (A0),d4               ; copia il byte dalla tabella in d4
                              ; in modo da farla trovare alla routine
                              ; universale

        ADDQ.L       #1,TABXPOINT
        MOVE.L       TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                              ; copiato in a0
        CMP.L       #FINETABX-1,A0
        BNE.S       NOBSTARTX
        MOVE.L       #TABX-1,TABXPOINT
NOBSTARTX:
        moveq       #0,d3               ; azzeriamo d3
        MOVE.b       (A0),d3        ; poniamo il valore della tabella in d3

        moveq       #15,d2               ; altezza dello sprite: e' la stessa per
                              ; tutti e 4, quindi la mettiamo in d2
                              ; una volta per tutte!

        lea        MIOSPRITE,A1        ; indirizzo sprite 0
        move.w       d4,d0               ; mettiamo le coordinate nei registri
        move.w       d3,d1
        bsr.w       UniMuoviSprite        ; esegue la routine universale che posiziona
                              ; lo sprite

        lea        MIOSPRITE1,A1        ; indirizzo sprite 1
        add.w       #16,d3               ; sprite 1 16 pixel piu' a destra di sprite 0
        move.w       d4,d0               ; mettiamo le coordinate nei registri
        move.w       d3,d1
        bsr.w       UniMuoviSprite        ; esegue la routine universale che posiziona
                              ; lo sprite

        lea        MIOSPRITE2,A1        ; indirizzo sprite 2
        add.w       #16,d3               ; sprite 2 16 pixel piu' a destra di sprite 1
        move.w       d4,d0               ; mettiamo le coordinate nei registri
        move.w       d3,d1
```

```
        bsr.w        UniMuoviSprite        ; esegue la routine universale che posiziona
                                ; lo sprite

        lea        MIOSPRITE3,A1        ; indirizzo sprite 3
        add.w        #16,d3                ; sprite 3 16 pixel piu‘ a destra di sprite 2
        move.w        d4,d0                ; mettiamo le coordinate nei registri
        move.w        d3,d1
        bsr.w        UniMuoviSprite        ; esegue la routine universale che posiziona
                                ; lo sprite
        rts


TABYPOINT:
        dc.l        TABY-1                ; NOTA: i valori della tabella qua sono bytes,
                                ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                                ; e non #2 come per quando sono word o con #4
                                ; come quando sono longword.
TABXPOINT:
        dc.l        TABX-1                ; NOTA: i valori della tabella qua sono byte

; Tabella con coordinate Y dello sprite precalcolate.
; Da notare che la posizione Y per far entrare lo sprite nella finestra video
; deve essere compresa tra $0 e $ff, infatti l’offset di $2c viene aggiunto
; dalla routine. Se non si usano schermi overscan, ossia non piu’ lunghi di
; 255 linee, si puo’ usare una tabella di valori dc.b (da $00 a $FF)

TABY:
        incbin        "ycoordinatok.tab"        ; 200 valori .B
FINETABY:


; Tabella con coordinate X dello sprite piu‘ a sinistra precalcolate.
; Questa tabella contiene valori reali, senza gli offset che sono aggiunti
; automaticamente dalla routine universale.
; Poiche‘ i 4 sprite insieme formano una figura larga 64 pixel, lo sprite
; piu‘ a sinistra puo‘ variare la sua posizione orizzontale tra 0 e
; 319-64=255. Questo fatto ci consente di usare anche per questa tabella
; dei bytes anziche‘ delle word
; La tabella e‘ fatta sempre con
; IS
; beg>0
; end>360
; amount>300
; amp>255/2
; y_offset>255/2
; multiplier>1

TABX:
        DC.B        $80,$83,$86,$88,$8B,$8E,$90,$93,$95,$98,$9B,$9D,$A0,$A2,$A5,$A8
        DC.B        $AA,$AD,$AF,$B1,$B4,$B6,$B9,$BB,$BD,$C0,$C2,$C4,$C6,$C9,$CB,$CD
        DC.B        $CF,$D1,$D3,$D5,$D7,$D9,$DB,$DC,$DE,$E0,$E2,$E3,$E5,$E7,$E8,$EA
        DC.B        $EB,$EC,$EE,$EF,$F0,$F1,$F2,$F4,$F5,$F6,$F6,$F7,$F8,$F9,$FA,$FA
        DC.B        $FB,$FB,$FC,$FC,$FD,$FD,$FD,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FD
        DC.B        $FD,$FD,$FC,$FC,$FB,$FB,$FA,$FA,$F9,$F8,$F7,$F6,$F6,$F5,$F4,$F2
        DC.B        $F1,$F0,$EF,$EE,$EC,$EB,$EA,$E8,$E7,$E5,$E3,$E2,$E0,$DE,$DC,$DB
        DC.B        $D9,$D7,$D5,$D3,$D1,$CF,$CD,$CB,$C9,$C6,$C4,$C2,$C0,$BD,$BB,$B9
        DC.B        $B6,$B4,$B1,$AF,$AD,$AA,$A8,$A5,$A2,$A0,$9D,$9B,$98,$95,$93,$90
        DC.B        $8E,$8B,$88,$86,$83,$80,$7E,$7B,$78,$76,$73,$70,$6E,$6B,$69,$66
        DC.B        $63,$61,$5E,$5C,$59,$56,$54,$51,$4F,$4D,$4A,$48,$45,$43,$41,$3E
        DC.B        $3C,$3A,$38,$35,$33,$31,$2F,$2D,$2B,$29,$27,$25,$23,$22,$20,$1E
        DC.B        $1C,$1B,$19,$17,$16,$14,$13,$12,$10,$0F,$0E,$0D,$0C,$0A,$09,$08
        DC.B        $08,$07,$06,$05,$04,$04,$03,$03,$02,$02,$01,$01,$01,$00,$00,$00
```

```
        DC.B        $00,$00,$00,$00,$00,$01,$01,$01,$02,$02,$03,$03,$04,$04,$05,$06
        DC.B        $07,$08,$08,$09,$0A,$0C,$0D,$0E,$0F,$10,$12,$13,$14,$16,$17,$19
        DC.B        $1B,$1C,$1E,$20,$22,$23,$25,$27,$29,$2B,$2D,$2F,$31,$33,$35,$38
        DC.B        $3A,$3C,$3E,$41,$43,$45,$48,$4A,$4D,$4F,$51,$54,$56,$59,$5C,$5E
        DC.B        $61,$63,$66,$69,$6B,$6E,$70,$73,$76,$78,$7B,$7E
FINETABX:



; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W       #$2c,d0                ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b       d0,(a1)                ; copia il byte in VSTART
        btst.l       #8,d0
        beq.s       NonVSTARTSET
        bset.b       #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s       ToVSTOP
NonVSTARTSET:
        bclr.b       #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w       D2,D0                ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b       d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l       #8,d0
        beq.s       NonVSTOPSET
        bset.b       #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w       VstopFIN
NonVSTOPSET:
        bclr.b       #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w       #128,D1                ; 128 - per centrare lo sprite.
        btst       #0,D1                ; bit basso della coordinata X azzerato?
        beq.s       BitBassoZERO
        bset       #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s       PlaceCoords

BitBassoZERO:
        bclr       #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w       #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b       D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts
```

```
        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

;        Palette della PIC

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

;        Palette degli SPRITE

        dc.w        $1A2,$800        ; color17
        dc.w        $1A4,$d00        ; color18
        dc.w        $1A6,$cc0        ; color19

         ; i colori per gli sprite 2 e 3 sono gli stessi che per sprite 0 e 1
        dc.w        $1AA,$800        ; color21
        dc.w        $1AC,$d00        ; color22
        dc.w        $1AE,$cc0        ; color23

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! **********

MIOSPRITE:                                ; lunghezza 15 linee
        incbin        "Largesprite0.raw"

MIOSPRITE1:                                ; lunghezza 15 linee
        incbin        "Largesprite1.raw"

MIOSPRITE2:                                ; lunghezza 15 linee
        incbin        "Largesprite2.raw"

MIOSPRITE3:                                ; lunghezza 15 linee
        incbin        "Largesprite3.raw"


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
```

```
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo listato utiliziamo 4 sprite a 4 colori per realizzare una figura
larga 64 pixel. Gli sprite sono allineati orizzontalmente. Pertanto hanno
tutti la stessa posizione verticale, mentre orizzontalmente distano 16 pixel
l'uno dall'altro.
Dalle tabelle leggiamo la posizione del primo sprite, mentre per gli altri
usiamo la stessa coordinata verticale e aggiungiamo ogni volta 16 pixel a
quella orizzontale.
Notate la comodita' di avere una routine universale: per spostare gli sprite
usiamo sempre la stessa routine, solo che ogni volta mettiamo un diverso
indirizzo in a1 e diverse coordinate nei registri d0 e d1. In questo caso
l'altezza e' sempre la stessa, per cui non modifichiamo d2. Se pero' avessimo
bisogno di muovere sprite di diverse altezze, non ci sarebbe problema,
basterebbe cambiare anche d2 e usare sempre la stessa routine universale.

Figura 23.5: Lezione 7p

23.15 Lezione7q

```
; Lezione7p.s        UNO SPRITE MOSSO CON IL JOYSTICK

        SECTION        CiriCop,CODE

Inizio:
```

```
        move.l      4.w,a6              ; Execbase
        jsr       -$78(a6)        ; Disable
        lea       GfxName(PC),a1        ; Nome lib
        jsr       -$198(a6)       ; OpenLibrary
        move.l      d0,GfxBase
        move.l      d0,a6
        move.l      $26(a6),OldCop        ; salviamo la vecchia COP

;       Puntiamo la PIC "vuota"

        MOVE.L      #BITPLANE,d0        ; dove puntare
        LEA       BPLPOINTERS,A1        ; puntatori COP
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)

;       Puntiamo lo sprite

        MOVE.L      #MIOSPRITE,d0               ; indirizzo dello sprite in d0
        LEA       SpritePointers,a1       ; Puntatori in copperlist
        move.w      d0,6(a1)
        swap       d0
        move.w      d0,2(a1)

        move.l      #COPPERLIST,$dff080       ; nostra COP
        move.w      d0,$dff088              ; START COP
        move.w      #0,$dff1fc              ; NO AGA!
        move.w      #$c00,$dff106              ; NO AGA!

mouse:
        cmpi.b      #$ff,$dff006        ; Linea 255?
        bne.s       mouse

        btst       #7,$bfe001       ; Tasto FIRE premuto?
        bne.s       NonFuoco       ; se no, salta l'istruzione seguente
        move.w      #$f00,$dff180       ; se si, il metti un bel ROSSO nel COLOR0
NonFuoco:

        bsr.s       LeggiJoyst       ; questa legge il joystick
        move.w      sprite_y(pc),d0 ; prepara i parametri per la routine
        move.w      sprite_x(pc),d1 ; universale
        lea       miosprite,a1       ; indirizzo sprite
        moveq       #13,d2              ; altezza sprite
        bsr.w       UniMuoviSprite       ; chiama la routine universale

Aspetta:
        cmpi.b      #$ff,$dff006        ; linea 255?
        beq.s       Aspetta

        btst       #6,$bfe001       ; mouse premuto?
        bne.s       mouse

        move.l      OldCop(PC),$dff080       ; Puntiamo la cop di sistema
        move.w      d0,$dff088              ; facciamo partire la vecchia cop

        move.l      4.w,a6
        jsr       -$7e(a6)        ; Enable
        move.l      gfxbase(PC),a1
        jsr       -$19e(a6)       ; Closelibrary
        rts

;       Dati
```

```
GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine legge il joystick e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiJoyst:
        MOVE.w       $dff00c,D3        ; JOY1DAT
        BTST.l       #1,D3               ; il bit 1 ci dice se si va a destra
        BEQ.S        NODESTRA        ; se vale zero non si va a destra
        ADDQ.w       #1,SPRITE_X        ; se vale 1 sposta a di un pixel lo sprite
        BRA.S        CHECK_Y               ; vai al controllo della Y
NODESTRA:
        BTST.l       #9,D3               ; il bit 9 ci dice se si va a sinistra
        BEQ.S        CHECK_Y               ; se vale zero non si va a sinistra
        SUBQ.W       #1,SPRITE_X        ; se vale 1 sposta lo sprite
CHECK_Y:
        MOVE.w       D3,D2               ; copia il valore del registro
        LSR.w        #1,D2               ; fa scorrere i bit di un posto verso destra
        EOR.w        D2,D3               ; esegue l'or esclusivo. Ora possiamo testare
        BTST.l       #8,D3               ; testiamo se va in alto
        BEQ.S        NOALTO              ; se no controlla se va in basso
        SUBQ.W       #1,SPRITE_Y        ; se si sposta lo sprite
        BRA.S        ENDJOYST
NOALTO:
        BTST.l       #0,D3               ; testiamo se va in basso
        BEQ.S        ENDJOYST        ; se no finisci
        ADDQ.W       #1,SPRITE_Y        ; se si sposta lo sprite
ENDJOYST:
        RTS

SPRITE_Y:       dc.w        0        ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.w        0        ; qui viene memorizzata la X dello sprite


; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W       #$2c,d0               ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b       d0,(a1)               ; copia il byte in VSTART
        btst.l       #8,d0
        beq.s        NonVSTARTSET
```

```
        bset.b      #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s       ToVSTOP
NonVSTARTSET:
        bclr.b      #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w       D2,D0               ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b      d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l      #8,d0
        beq.s       NonVSTOPSET
        bset.b      #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w       VstopFIN
NonVSTOPSET:
        bclr.b      #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w       #128,D1               ; 128 - per centrare lo sprite.
        btst        #0,D1               ; bit basso della coordinata X azzerato?
        beq.s       BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s       PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w       #1,D1               ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b      D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts



        SECTION     GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81       ; DiwStrt
        dc.w        $90,$2cc1       ; DiwStop
        dc.w        $92,$38             ; DdfStart
        dc.w        $94,$d0             ; DdfStop
        dc.w        $102,0              ; BplCon1
        dc.w        $104,0              ; BplCon2
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000      ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo      bitplane

        dc.w        $180,$000       ; color0        ; sfondo nero
        dc.w        $182,$123       ; color1        ; colore 1 del bitplane, che
                                        ; in questo caso e' vuoto,
```

```
                                        ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit

 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0        ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo esempio muoviamo uno sprite con il joystick.
La cosa piu' facile e' rilevare se il bottone FIRE e' premuto, infatti basta
un BTST #7,$bfe001, analogamente al tasto sinistro del mouse che e' il bit 6.
Per posizionare lo sprite sullo schermo usiamo la nostra routine universale,
che abbiamo gia‘ bella pronta risparmiandoci un po' di lavoro.
La routine Leggijoyst invece si occupa di rilevare lo stato del joystick e di
conseguenza aggiorna le coordinate dello sprite che sono memorizzate
in 2 locazioni di memoria: SPRITE_X e SPRITE_Y. Per leggere il joystick
e‘ necessario utilizzare l'istruzione EOR che, come abbiamo visto nella lezione
esegue un'operazione di OR ESCLUSIVO tra i bit di 2 registri.
Infatti la lettura del joystick avviene tramite il registro JOY1DAT. Per sapere
se la leva del joystick e‘ stata premuta a destra o a sinistra, basta sapere
lo stato dei bit 1 e 9. per le altre direzioni e‘ un po' piu‘ complicato.
Infatti, per sapere se la leva del joystick e‘ spinta verso l'alto bisogna
calcolare l'OR ESCLUSIVO tra il bit 8 e il bit 9 del registro JOY1DAT.
Poiche‘ questi 2 bit si trovano sullo stesso registro, prima copiamo il

registro in 2 registri dati del 68000, per esempio D2 e D3. Poi SHIFTIAMO
(cioe‘ facciamo scorrere) verso destra i bit di uno dei 2 registri dati.
In questo modo il bit 9 del registro dati viene spostato nella posizione 8.
Poiche‘ il registro che abbiamo SHIFTATO conteneva una copia di JOY1DAT,
dopo lo SHIFT il bit 8 del registro dati sara‘ uguale al bit 9 di JOY1DAT.
Nel registro non SHIFTATO, invece il bit 8 e‘ uguale al bit 8 di JOY1DAT.
Facendo ora l’EOR tra i due registri, nella posizione 8 ci sara‘ quindi l’EOR
tra il bit 8 del registro JOY1DAT e il bit 9 del registro JOY1DAT. Proprio
quello che ci serviva per sapere se dobbiamo muovere lo sprite verso l’alto.
Per quanto riguarda il basso si deve calcolare l’OR ESCLUSIVO tra i bit 0 e 1
nello stesso modo che per l’alto.

Potete provare a variare la velocita‘ dello sprite. Nella routine LeggiJoyst
quando viene rilevato che la leva e‘ stata spostata in una certa direzione
viene corrrispondentemente spostato di 1 pixel lo sprite con una ADDQ #1,xxx
(o con una SUBQ #1,xxx) . Se invece di 1 mettete valori maggiori lo sprite
si muovera‘ piu‘ velocemente.

23.16 Lezione7r

```
; Lezione7r1.s        UNO SPRITE MOSSO CON IL MOUSE

;               NOTA: Questa routine orizzontalmente puo' gestire solo
;                     255 linee e non tutte le 320.


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1        ; Nome lib
        jsr         -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA         BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
```

```
        bne.s       mouse

        bsr.s       LeggiMouse        ; questa legge il mouse
        moveq       #0,d0               ; pulisci d0
        moveq       #0,d1               ; pulisci d1

; prepara i parametri per la routine universale

        move.b       sprite_y(pc),d0 ; coordinate sprite
        move.b       sprite_x(pc),d1
        lea        miosprite,a1        ; indirizzo sprite
        moveq       #13,d2               ; altezza sprite
        bsr.s       UniMuoviSprite        ; chiama la routine universale

Aspetta:
        cmpi.b       #$ff,$dff006        ; linea 255?
        beq.s       Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s       mouse

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b       $dff00a,d0      ; JOY0DAT posizione verticale mouse
        move.b       d0,sprite_y        ; modifica posizione sprite

        move.b       $dff00b,d0        ; JOY0DAT+1 posizione orizzontale mouse
        move.b       d0,sprite_x        ; modifica pos. sprite
        RTS

SPRITE_Y:       dc.b        0        ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.b        0        ; qui viene memorizzata la X dello sprite



; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
```

```
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W        #$2c,d0                ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b        d0,(a1)                ; copia il byte in VSTART
        btst.l        #8,d0
        beq.s        NonVSTARTSET
        bset.b        #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s        ToVSTOP
NonVSTARTSET:
        bclr.b        #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w        D2,D0                ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l        #8,d0
        beq.s        NonVSTOPSET
        bset.b        #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w        VstopFIN
NonVSTOPSET:
        bclr.b        #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w        #128,D1                ; 128 - per centrare lo sprite.
        btst        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
```

```
        dc.w        $102,0              ; BplCon1
        dc.w        $104,0              ; BplCon2
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000      ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000       ; color0        ; sfondo nero
        dc.w        $182,$123       ; color1        ; colore 1 del bitplane, che
                                                    ; in questo caso e' vuoto,
                                                    ; per cui non compare.

        dc.w        $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE       ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit

 dc.w       %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w       %0000000000000000,%0000011001100000
 dc.w       %0000000000000000,%0000001001000000
 dc.w       %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w       %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w       %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w       %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w       %0000011111100000,%0000011111100000
 dc.w       %0000011111100000,%0001111001111000
 dc.w       %0000001111000000,%0011101111011100
 dc.w       %0000000110000000,%0011000110001100
 dc.w       %0000000000000000,%1111000000001111
 dc.w       %0000000000000000,%1111000000001111
 dc.w       0,0         ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                            ; perche' per vedere gli sprite
                                            ; e' necessario che ci siano bitplanes
                                            ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo esempio muoviamo uno sprite con il mouse.
Per posizionare lo sprite sullo schermo usiamo la nostra routine universale,

che abbiamo gia‘ bella pronta risparmiandoci un po’ di lavoro.
La routine LeggiMouse invece si occupa di rilevare lo stato del mouse e di
conseguenza aggiorna le coordinate dello sprite che sono memorizzate
in 2 locazioni di memoria: SPRITE_X e SPRITE_Y.

La lettura del mouse avviene separatamente per le posizioni orizzontali e
verticali. Il registro JOY0DAT lo possiamo considerare come una coppia di
byte. Il byte all’indirizzo JOY0DAT (=$dff00a) si occupa della posizione
verticale, mentre quello all’indirizzo JOY0DAT+1 (=$dff00b) si occupa
della posizione orizzontale. In ogni byte leggiamo un numero, che va
da 0 a 255, che varia a seconda dei movimenti del mouse:

alto - $dff00a decresce
basso - $dff00a cresce
sinistra - $dff00b decresce
destra - $dff00b cresce

In questo esempio usiamo i numeri che leggiamo dal mouse direttamente
come coordinate dello sprite.
Potete subito vedere come questo semplice metodo, ha un "piccolo" problema:
i numeri che leggiamo dal mouse vanno da 0 a 255, mentre le coordinate
orizzontali vanno da 0 a 320. Quindi con questo metodo lo sprite non puo‘
raggiungere il bordo destro dello schermo. Inoltre lo stesso problema si
presenterebbe per le coordinate verticali se usassimo uno schermo in
overscan.
Come risolverlo ? Lo vedremo nel prossimo esempio.

Lezione7r2

```
; Lezione7r2.s        UNO SPRITE MOSSO CON IL MOUSE CHE ARRIVA FINO AL BORDO DESTRO



        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
```

```
        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                 ; START COP
        move.w        #0,$dff1fc                 ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

        move.b        $dff00a,mouse_y
        move.b        $dff00b,mouse_x

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        LeggiMouse        ; questa legge il mouse
        move.w        sprite_y(pc),d0 ; prepara i parametri per la routine
        move.w        sprite_x(pc),d1 ; universale
        lea        miosprite,a1        ; indirizzo sprite
        moveq        #13,d2                ; altezza sprite
        bsr.w        UniMuoviSprite        ; chiama la routine universale

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b        $dff00a,d1        ; JOY0DAT posizione verticale mouse
        move.b        d1,d0                ; copia in d0
        sub.b        mouse_y(PC),d0        ; sottrai vecchia posizione mouse
        beq.s        no_vert                ; se la differenza = 0, il mouse e‘ fermo
        ext.w        d0                ; trasforma il byte in word
                            ; (vedi alla fine del listato)
        add.w        d0,sprite_y        ; modifica posizione sprite
no_vert:
          move.b        d1,mouse_y        ; salva posizione mouse per la prossima volta

        move.b        $dff00b,d1        ; posizione orizzontale mouse
        move.b        d1,d0                ; copia in d0
```

```
        sub.b        mouse_x(PC),d0        ; sottrai vecchia posizione
        beq.s        no_oriz                ; se la differenza = 0, il mouse e‘ fermo
        ext.w        d0                ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w        d0,sprite_x        ; modifica pos. sprite
no_oriz
          move.b        d1,mouse_x        ; salva posizione mouse per la prossima volta
        RTS

SPRITE_Y:        dc.w        0        ; qui viene memorizzata la Y dello sprite
SPRITE_X:        dc.w        0        ; qui viene memorizzata la X dello sprite
MOUSE_Y:        dc.b        0        ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b        0        ; qui viene memorizzata la X del mouse



; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W        #$2c,d0                ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b        d0,(a1)                ; copia il byte in VSTART
        btst.l        #8,d0
        beq.s        NonVSTARTSET
        bset.b        #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s        ToVSTOP
NonVSTARTSET:
        bclr.b        #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w        D2,D0                ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l        #8,d0
        beq.s        NonVSTOPSET
        bset.b        #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w        VstopFIN
NonVSTOPSET:
        bclr.b        #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w        #128,D1                ; 128 - per centrare lo sprite.
        btst        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
```

```
        lsr.w        #1,D1               ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                  ; il valore di HSTART, per "trasformarlo" nel
                                  ; valore fa porre nel byte HSTART, senza cioe'
                                  ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                          ; in questo caso e' vuoto,
                                          ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE:                ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit

 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
```

```
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0         ; 2 word azzerate definiscono la fine dello sprite.


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                 ; bitplane azzerato lowres

        end
```

In questo esempio muoviamo uno sprite con il mouse in modo da raggiungere
il bordo destro e da non avere problemi con un eventuale overscan verticale.

Se vogliamo raggiungere il bordo destro dobbiamo usare una word per la
posizione orizzontale dello sprite. Il mouse pero‘ ci fornisce delle
coordinate in forma di byte. Allora usiamo il seguente metodo:
memorizziamo separatamente le coordinate dello sprite e le coordinate fornite
dal mouse. Ogni volta che eseguiamo LeggiMouse, leggiamo delle nuove
coordinate e le confrontiamo con le vecchie. Calcoliamo la differenza tra le
vecchie e le nuove coordiante del mouse, e aggiungiamo questa differenza alle
coordinate dello sprite. In questo modo non ha importanza il fatto che quando
la posizione del mouse supera 255 ritorna a 0, perche‘ cio‘ che conta e‘ solo
la differenza tra la nuova e la vecchia coordinata. Vi ricordo infatti che se
un byte assume un valore da 128 a 255, quando lo usiamo in un’addizione o in
una sottrazione viene considerato un numero negativo in complemento a due.
Per cui se la vecchia coordinata vale 255, e la nuova vale 1, facendo la
differenza 255(=$ff) viene considerato -1. Quindi 1-(-1)=2.
Questo numero 2 viene aggiunto alla coordinata x dello sprite e siccome e‘
positivo provoca comunque uno spostamento verso destra.
Se invece la differenza fosse stata negativa, aggiungendola alla coordinata x
dello sprite avrebbe provocato uno spostamento verso sinistra.
C’e‘ comunque un particolare a cui bisogna prestare molta attenzione. Quando
facciamo la differenza tra le coordinate del mouse stiamo lavorando con due
byte. Per cui la differenza sara‘ ancora un byte. Questo byte poi lo sommiamo
alla coordinata dello sprite che e‘ una word. Cio‘ provoca un problema.
Prima di fare la somma e‘ necessario trasformare il byte in una word.
La trasformazione viene fatta dal’istruzione EXT che trasforma un byte
contenuto in un registro in una word. Vediamo come opera tale istruzione.
Ci sono 2 casi:
Il byte contiene un numero positivo, per es. 5. La EXT trasforma cosi‘:

```
        Contenuto prima della EXT        Contenuto dopo la EXT
        $XX05                                $0005
(XX indica un qualsiasi numero)
```

Infatti 5 in formato word si scrive proprio $0005

Il byte contiene un numero negativo, per es. -5. La EXT trasforma cosi‘:
Ricordando che -5 in formato byte si scrive $FB

```
        Contenuto prima della EXT        Contenuto dopo la EXT
        $XXFB                                $FFFB
(XX indica un qualsiasi numero)
```

```
Infatti -5 in formato word si scrive proprio $FFFB.

In pratica la EXT prende il bit 7 di un registro (il bit che indica il segno)
e lo copia nei bit da 8 a 15.
Anche se non e' usata in questo esempio sappiate che per trasformare una word
in una long-word si usa sempre l'istruzione EXT, solo in formato .L:

        EXT.L   d0        ; trasforma una word in long-word

La trasformazione avviene nella stessa maniera.

Per quanto riguarda le posizioni verticali il discorso e' lo stesso, e
infatti la rouine e' identica.

Per posizionare lo sprite sullo schermo usiamo di nuovo la routine universale,
che abbiamo gia' bella pronta. Vi renderete conto che anche le routine che
gestiscono la letture del mouse e del joystick si possono usare in ogni
programma che sia gestito il joystick e il mouse. Infatti i programmatori di
giochi e demo riutilizzano gran parte delle routines.
```

23.17 Lezione7s

```
; Lezione7s.s        VISUALIZZAZIONE DI 16 SPRITE

;        In questo listato si mostra come riutilizzare gli sprite.
;        Premendo il tasto sinistro gli sprite cambiano posizione.
;        Tasto destro del mouse per uscire.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1        ; Nome lib
        jsr         -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop


        MOVE.L        #BITPLANE,d0
        LEA         BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo gli sprite

        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE1,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE2,d0                ; indirizzo dello sprite in d0
```

```
        addq.w      #8,a1                       ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE3,d0               ; indirizzo dello sprite in d0
        addq.w      #8,a1                       ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE4,d0               ; indirizzo dello sprite in d0
        addq.w      #8,a1                       ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE5,d0               ; indirizzo dello sprite in d0
        addq.w      #8,a1                       ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE6,d0               ; indirizzo dello sprite in d0
        addq.w      #8,a1                       ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)
        MOVE.L      #MIOSPRITE7,d0               ; indirizzo dello sprite in d0
        addq.w      #8,a1                       ; prossimi SPRITEPOINTERS
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

; posizioni sprite
        MOVE.B      #$2C+50,VSTART0
        MOVE.B      #$2C+50+8,VSTOP0
        MOVE.B      #$2C+50,VSTART1
        MOVE.B      #$2C+50+8,VSTOP1
        MOVE.B      #$2C+50,VSTART2
        MOVE.B      #$2C+50+8,VSTOP2
        MOVE.B      #$2C+50,VSTART3
        MOVE.B      #$2C+50+8,VSTOP3
        MOVE.B      #$2C+50,VSTART4
        MOVE.B      #$2C+50+8,VSTOP4
        MOVE.B      #$2C+50,VSTART5
        MOVE.B      #$2C+50+8,VSTOP5
        MOVE.B      #$2C+50,VSTART6
        MOVE.B      #$2C+50+8,VSTOP6
        MOVE.B      #$2C+50,VSTART7
        MOVE.B      #$2C+50+8,VSTOP7

; da qui iniziano gli sprite "riusati"
        MOVE.B      #$2C+90,VSTART8
        MOVE.B      #$2C+90+8,VSTOP8
        MOVE.B      #$2C+90,VSTART9
        MOVE.B      #$2C+90+8,VSTOP9
        MOVE.B      #$2C+90,VSTART10
        MOVE.B      #$2C+90+8,VSTOP10
        MOVE.B      #$2C+90,VSTART11
        MOVE.B      #$2C+90+8,VSTOP11
        MOVE.B      #$2C+90,VSTART12
        MOVE.B      #$2C+90+8,VSTOP12
        MOVE.B      #$2C+90,VSTART13
        MOVE.B      #$2C+90+8,VSTOP13
        MOVE.B      #$2C+90,VSTART14
```

```
        MOVE.B        #$2C+90+8,VSTOP14
        MOVE.B        #$2C+90,VSTART15
        MOVE.B        #$2C+90+8,VSTOP15


        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

Mouse1:
        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse1

; mette nuove posizioni verticali

        MOVE.B        #$2C+10,VSTART0
        MOVE.B        #$2C+10+8,VSTOP0
        MOVE.B        #$2C+10+8*1,VSTART1
        MOVE.B        #$2C+10+8*1+8,VSTOP1
        MOVE.B        #$2C+10+8*2,VSTART2
        MOVE.B        #$2C+10+8*2+8,VSTOP2
        MOVE.B        #$2C+10+8*3,VSTART3
        MOVE.B        #$2C+10+8*3+8,VSTOP3
        MOVE.B        #$2C+10+8*4,VSTART4
        MOVE.B        #$2C+10+8*4+8,VSTOP4
        MOVE.B        #$2C+10+8*5,VSTART5
        MOVE.B        #$2C+10+8*5+8,VSTOP5
        MOVE.B        #$2C+10+8*6,VSTART6
        MOVE.B        #$2C+10+8*6+8,VSTOP6
        MOVE.B        #$2C+10+8*7,VSTART7
        MOVE.B        #$2C+10+8*7+8,VSTOP7

; da qui iniziano gli sprite "riusati"

        MOVE.B        #$2C+10+20,VSTART8
        MOVE.B        #$2C+10+20+8,VSTOP8
        MOVE.B        #$2C+10+20+8*1,VSTART9
        MOVE.B        #$2C+10+20+8*1+8,VSTOP9
        MOVE.B        #$2C+10+20+8*2,VSTART10
        MOVE.B        #$2C+10+20+8*2+8,VSTOP10
        MOVE.B        #$2C+10+20+8*3,VSTART11
        MOVE.B        #$2C+10+20+8*3+8,VSTOP11
        MOVE.B        #$2C+10+20+8*4,VSTART12
        MOVE.B        #$2C+10+20+8*4+8,VSTOP12
        MOVE.B        #$2C+10+20+8*5,VSTART13
        MOVE.B        #$2C+10+20+8*5+8,VSTOP13
        MOVE.B        #$2C+10+20+8*6,VSTART14
        MOVE.B        #$2C+10+20+8*6+8,VSTOP14
        MOVE.B        #$2C+10+20+8*7,VSTART15
        MOVE.B        #$2C+10+20+8*7+8,VSTOP15

Mouse2:
        btst        #2,$dff016
        bne.s        Mouse2


        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
```

```
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$F00        ; color17, - COLOR1 degli sprite0/1 -ROSSO
        dc.w        $1A4,$0F0        ; color18, - COLOR2 degli sprite0/1 -VERDE
        dc.w        $1A6,$FF0        ; color19, - COLOR3 degli sprite0/1 -GIALLO

        dc.w        $1AA,$FFF        ; color21, - COLOR1 degli sprite2/3 -BIANCO
        dc.w        $1AC,$0BD        ; color22, - COLOR2 degli sprite2/3 -ACQUA
        dc.w        $1AE,$D50        ; color23, - COLOR3 degli sprite2/3 -ARANCIO

        dc.w        $1B2,$00F        ; color25, - COLOR1 degli sprite4/5 -BLU
        dc.w        $1B4,$F0F        ; color26, - COLOR2 degli sprite4/5 -VIOLA
        dc.w        $1B6,$BBB        ; color27, - COLOR3 degli sprite4/5 -GRIGIO

        dc.w        $1BA,$8E0        ; color29, - COLOR1 degli sprite6/7 -VERDE CH.
        dc.w        $1BC,$a70        ; color30, - COLOR2 degli sprite6/7 -MARRONE
        dc.w        $1BE,$d00        ; color31, - COLOR3 degli sprite6/7 -ROSSO SC.

        dc.w        $FFFF,$FFFE        ; Fine della copperlist
```

```
; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! ************

 ; tabella di riferimento per definire i colori:


 ; per gli sprite 0 ed 1
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (ROSSO)
 ;BINARIO 01=COLORE 2 (VERDE)
 ;BINARIO 11=COLORE 3 (GIALLO)

MIOSPRITE0:                ; lunghezza 13 linee
VSTART0:
        dc.b 0
HSTART0:
        dc.b $40+12+0*20
VSTOP0:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
VSTART8:
        dc.b $0
HSTART8:
        dc.b $40+20+0*12
VSTOP8:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000001110000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite


MIOSPRITE1:                ; lunghezza 13 linee
VSTART1:
        dc.b $0
HSTART1:
        dc.b $40+12+1*20
VSTOP1:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000010001111
 dc.w        %0011111111111100,%1100000110000011
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0011111111111100,%1100000010000011
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0000001111000000,%0111110000111110
```

```
VSTART9:
        dc.b $0
HSTART9:
        dc.b $40+20+1*12
VSTOP9:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000001110000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

 ; per gli sprite 2 e 3
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BIANCO)
 ;BINARIO 01=COLORE 2 (ACQUA)
 ;BINARIO 11=COLORE 3 (ARANCIO)

MIOSPRITE2:                ; lunghezza 13 linee
VSTART2:
        dc.b $0
HSTART2:
        dc.b $40+12+2*20
VSTOP2:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0011111111111100,%1100001000100011
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0111111111111110,%1000000111000001
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
VSTART10:
        dc.b $0
HSTART10:
        dc.b $40+20+2*12
VSTOP10:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0011111111111100,%1100010011100011
 dc.w        %0111111111111110,%1000110010100001
 dc.w        %0111111111111110,%1000010010100001
 dc.w        %0011111111111100,%1100111011100011
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

MIOSPRITE3:                ; lunghezza 13 linee
VSTART3:
        dc.b $0
HSTART3:
        dc.b $40+12+3*20
VSTOP3:
```

```
        dc.b 0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0111111111111110,%1000000111100001
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
VSTART11:
        dc.b $0
HSTART11:
        dc.b $40+20+3*12
VSTOP11:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000110011000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100111011100011
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

 ; per gli sprite 4 e 5
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BLU)
 ;BINARIO 01=COLORE 2 (VIOLA)
 ;BINARIO 11=COLORE 3 (GRIGIO)

MIOSPRITE4:                ; lunghezza 13 linee
VSTART4:
        dc.b $0
HSTART4:
        dc.b $40+12+4*20
VSTOP4:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001001001111
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
 dc.w        %0000001111000000,%0111110000111110
VSTART12:
        dc.b $0
HSTART12:
        dc.b $40+20+4*12
VSTOP12:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0011111111111100,%1100010011000011
 dc.w        %0111111111111110,%1000110001000001
 dc.w        %0111111111111110,%1000010010000001
 dc.w        %0011111111111100,%1100111011100011
```

```
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

MIOSPRITE5:                 ; lunghezza 13 linee
VSTART5:
        dc.b $0
HSTART5:
        dc.b $40+12+5*20
VSTOP5:
        dc.b $0
        dc.b $0
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
VSTART13:
        dc.b $0
HSTART13:
        dc.b $40+20+5*12
VSTOP13:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0011111111111100,%1100010011100011
 dc.w        %0111111111111110,%1000110001100001
 dc.w        %0111111111111110,%1000010000100001
 dc.w        %0011111111111100,%1100111011000011
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

 ; per gli sprite 6 e 7
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (VERDE CHIARO)
 ;BINARIO 01=COLORE 2 (MARRONE)
 ;BINARIO 11=COLORE 3 (ROSSO SCURO)

MIOSPRITE6:                 ; lunghezza 13 linee
VSTART6:
        dc.b $0
HSTART6:
        dc.b $40+12+6*20
VSTOP6:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000001001000001
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
VSTART14:
        dc.b $0
HSTART14:
```

```
        dc.b $40+20+6*12
VSTOP14:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0011111111111100,%1100010000100011
 dc.w        %0111111111111110,%1000110010100001
 dc.w        %0111111111111110,%1000010011100001
 dc.w        %0011111111111100,%1100111001000011
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

MIOSPRITE7:                 ; lunghezza 13 linee
VSTART7:
        dc.b 0
HSTART7:
        dc.b $40+12+7*20
VSTOP7:
        dc.b $0
        dc.b $0
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
 dc.w        %0000001111000000,%0111110000111110
VSTART15:
        dc.b $0
HSTART15:
        dc.b $40+20+7*12
VSTOP15:
        dc.b $0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0011111111111100,%1100010011100011
 dc.w        %0111111111111110,%1000110011000001
 dc.w        %0111111111111110,%1000010000100001
 dc.w        %0011111111111100,%1100111011100011
 dc.w        %0000111111110000,%1111000000001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite

        SECTION         PLANEVUOTO,BSS_C
BITPLANE:
        ds.b        40*256

        end
```

In questo listato viene mostrato come riutilizzare gli sprite piu‘ volte sulla stessa schermata. Nell’esempio, ogni sprite viene utilizzato due volte.
Lo sprite 0 viene riutilizzato per disegnare lo sprite 8.
Lo sprite 1 viene riutilizzato per disegnare lo sprite 9.
Lo sprite 2 viene riutilizzato per disegnare lo sprite 10.
Lo sprite 3 viene riutilizzato per disegnare lo sprite 11.
Lo sprite 4 viene riutilizzato per disegnare lo sprite 12.
Lo sprite 5 viene riutilizzato per disegnare lo sprite 13.
Lo sprite 6 viene riutilizzato per disegnare lo sprite 14.

```
Lo sprite 7 viene riutilizzato per disegnare lo sprite 15.

Notate che quando uno sprite viene utilizzato la seconda volta, esso viene
posizionato sullo schermo PIU‘ IN BASSO dell’ultima riga dello sprite
visualizzato durante il primo utilizzo. Questo e‘ dovuto ad una precisa limita-
zione dell’hardware. Infatti tra un utilizzo e il successivo di uno sprite e‘
necessario lasciare ALMENO una riga vuota.

Il byte VSTART dello sprite 8 deve essere MAGGIORE di VSTOP dello sprite 0
Il byte VSTART dello sprite 9 deve essere MAGGIORE di VSTOP dello sprite 1
Il byte VSTART dello sprite 10 deve essere MAGGIORE di VSTOP dello sprite 2
Il byte VSTART dello sprite 11 deve essere MAGGIORE di VSTOP dello sprite 3
Il byte VSTART dello sprite 12 deve essere MAGGIORE di VSTOP dello sprite 4
Il byte VSTART dello sprite 13 deve essere MAGGIORE di VSTOP dello sprite 5
Il byte VSTART dello sprite 14 deve essere MAGGIORE di VSTOP dello sprite 6
Il byte VSTART dello sprite 15 deve essere MAGGIORE di VSTOP dello sprite 7

Riutilizzare uno sprite non cambiano i registri colore che ad esso sono
assegnati.
Nell’esempio potete notare infatti che uno sprite "riusato" ha gli stessi
colori di quelli "originali". Poiche‘ pero‘ gli sprite sono posizionati
sullo schermo a diverse altezze, nulla ci impedisce di cambiare i valori dei
registri colore tra un utilizzo e l’altro usando il copper. Potete farlo
per esercizio.
```

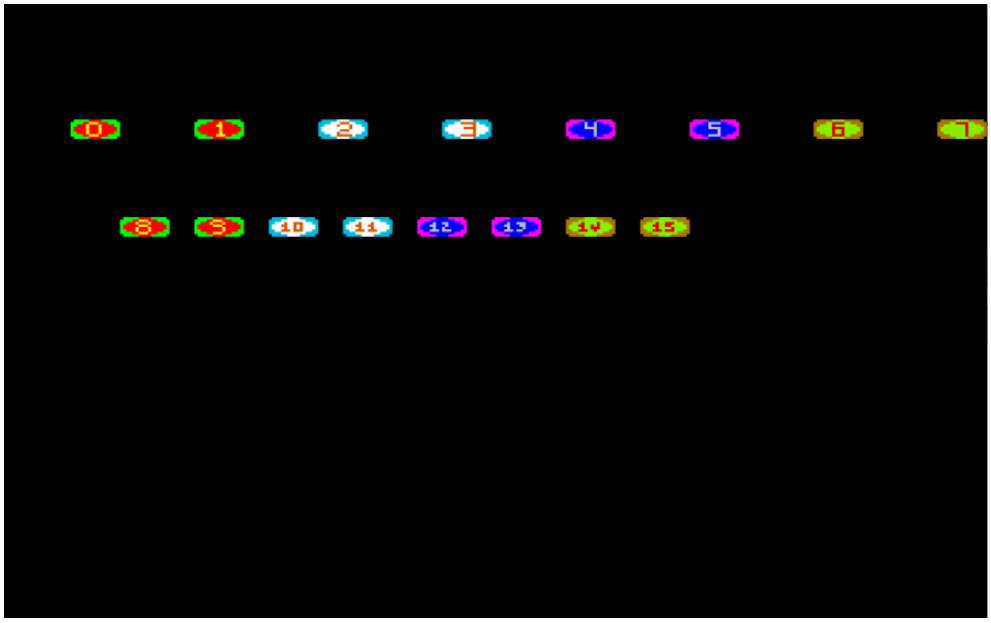


Figura 23.6: Lezione 7s

23.18 Lezione7t

```
; Lezione7t1.s        stelle
```

```
;        In questo listato facciamo un cielo stellato
;        con uno sprite riutilizzato 127 volte !!!

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop


        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo gli sprite

        MOVE.L        #SPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviStelle        ; questa routine muove le stelle

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
```

```
        dc.l        0

OldCop:
        dc.l        0

; Questa semplice routine fa avanzare lo sprite che costituisce le stelle
; agendo su ogni HSTART tramite il loop STELLE_LOOP

MuoviStelle:
        lea        Sprite,A0               ; a0 punta allo sprite
STELLE_LOOP:
        CMPI.B       #$f0,1(A0)               ; la stella ha raggiunto il bordo
                                        ; destro dello schermo ?
        BNE.S       no_bordo                ; no, allora salta
        MOVE.B       #$30,1(A0)               ; si, rimetti la stella a sinistra
no_bordo:
        ADDQ.B  #1,1(A0)                ; sposta di 2 pixel lo sprite
        ADDQ.w       #8,A0                      ; prossima stella
        CMP.L       #SpriteEnd,A0               ; abbiamo raggiunto la fine?
        BLO.S       STELLE_LOOP               ; se no rifai il loop
        RTS                                ; fine routine


        SECTION       GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop

        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$000        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

; lo sprite usa un solo colore, il 17

        dc.w        $1A2,$ddd        ; color17, - COLOR1 dello sprite0 - bianco

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        -        -        -        -        -        -        -

; Come potete notare, questo sprite e' riutilizzato un bel po' di volte, per
```

```
; la precisione 127. E' composto di tante coppie di word di controllo seguite
; da un $1000,$0000, ossia la linea di dati che compone ogni singola stella.
; Le posizioni verticali vanno di 2 in 2, per cui c'e' una stella ogni 2
; linee verticali, ad una posizione orizzontale diversa per ogni stella.
; il primo utilizzo dello sprite e' $307a,$3100,$1000,$0000
; il secondo e' $3220,$3300,$1000,$0000
; E cosi' via. I vari VSTART vanno di 2 in 2, infatti sono $30,$32,$34,$36...
; Gli HSTART sono posizionati casualmente ($7a,$20,$c0,$50...)
; I Vstop sono chiaramente 1 linea dopo lo start ($31,$33,$35..) essendo le
; stelle alte 1 pixel.
; La routine agisce ogni fotogramma su tutti gli HSTART spostando avanti le
; stelle.

Sprite:
        dc.w    $307A,$3100,$1000,$0000,$3220,$3300,$1000,$0000
        dc.w    $34C0,$3500,$1000,$0000,$3650,$3700,$1000,$0000
        dc.w    $3842,$3900,$1000,$0000,$3A6D,$3B00,$1000,$0000
        dc.w    $3CA2,$3D00,$1000,$0000,$3E9C,$3F00,$1000,$0000
        dc.w    $40DA,$4100,$1000,$0000,$4243,$4300,$1000,$0000
        dc.w    $445A,$4500,$1000,$0000,$4615,$4700,$1000,$0000
        dc.w    $4845,$4900,$1000,$0000,$4A68,$4B00,$1000,$0000
        dc.w    $4CB8,$4D00,$1000,$0000,$4EB4,$4F00,$1000,$0000
        dc.w    $5082,$5100,$1000,$0000,$5292,$5300,$1000,$0000
        dc.w    $54D0,$5500,$1000,$0000,$56D3,$5700,$1000,$0000
        dc.w    $58F0,$5900,$1000,$0000,$5A6A,$5B00,$1000,$0000
        dc.w    $5CA5,$5D00,$1000,$0000,$5E46,$5F00,$1000,$0000
        dc.w    $606A,$6100,$1000,$0000,$62A0,$6300,$1000,$0000
        dc.w    $64D7,$6500,$1000,$0000,$667C,$6700,$1000,$0000
        dc.w    $68C4,$6900,$1000,$0000,$6AC0,$6B00,$1000,$0000
        dc.w    $6C4A,$6D00,$1000,$0000,$6EDA,$6F00,$1000,$0000
        dc.w    $70D7,$7100,$1000,$0000,$7243,$7300,$1000,$0000
        dc.w    $74A2,$7500,$1000,$0000,$7699,$7700,$1000,$0000
        dc.w    $7872,$7900,$1000,$0000,$7A77,$7B00,$1000,$0000
        dc.w    $7CC2,$7D00,$1000,$0000,$7E56,$7F00,$1000,$0000
        dc.w    $805A,$8100,$1000,$0000,$82CC,$8300,$1000,$0000
        dc.w    $848F,$8500,$1000,$0000,$8688,$8700,$1000,$0000
        dc.w    $88B9,$8900,$1000,$0000,$8AAF,$8B00,$1000,$0000
        dc.w    $8C48,$8D00,$1000,$0000,$8E68,$8F00,$1000,$0000
        dc.w    $90DF,$9100,$1000,$0000,$924F,$9300,$1000,$0000
        dc.w    $9424,$9500,$1000,$0000,$96D7,$9700,$1000,$0000
        dc.w    $9859,$9900,$1000,$0000,$9A4F,$9B00,$1000,$0000
        dc.w    $9C4A,$9D00,$1000,$0000,$9E5C,$9F00,$1000,$0000
        dc.w    $A046,$A100,$1000,$0000,$A2A6,$A300,$1000,$0000
        dc.w    $A423,$A500,$1000,$0000,$A6FA,$A700,$1000,$0000
        dc.w    $A86C,$A900,$1000,$0000,$AA44,$AB00,$1000,$0000
        dc.w    $AC88,$AD00,$1000,$0000,$AE9A,$AF00,$1000,$0000
        dc.w    $B06C,$B100,$1000,$0000,$B2D4,$B300,$1000,$0000
        dc.w    $B42A,$B500,$1000,$0000,$B636,$B700,$1000,$0000
        dc.w    $B875,$B900,$1000,$0000,$BA89,$BB00,$1000,$0000
        dc.w    $BC45,$BD00,$1000,$0000,$BE24,$BF00,$1000,$0000
        dc.w    $C0A3,$C100,$1000,$0000,$C29D,$C300,$1000,$0000
        dc.w    $C43F,$C500,$1000,$0000,$C634,$C700,$1000,$0000
        dc.w    $C87C,$C900,$1000,$0000,$CA1D,$CB00,$1000,$0000
        dc.w    $CC6B,$CD00,$1000,$0000,$CEAC,$CF00,$1000,$0000
        dc.w    $D0CF,$D100,$1000,$0000,$D2FF,$D300,$1000,$0000
        dc.w    $D4A5,$D500,$1000,$0000,$D6D6,$D700,$1000,$0000
        dc.w    $D8EF,$D900,$1000,$0000,$DAE1,$DB00,$1000,$0000
        dc.w    $DCD9,$DD00,$1000,$0000,$DEA6,$DF00,$1000,$0000
        dc.w    $E055,$E100,$1000,$0000,$E237,$E300,$1000,$0000
        dc.w    $E47D,$E500,$1000,$0000,$E62E,$E700,$1000,$0000
        dc.w    $E8AF,$E900,$1000,$0000,$EA46,$EB00,$1000,$0000
        dc.w        $EC65,$ED00,$1000,$0000,$EE87,$EF00,$1000,$0000
```

```
        dc.w        $F0D4,$F100,$1000,$0000,$F2F5,$F300,$1000,$0000
        dc.w        $F4FA,$F500,$1000,$0000,$F62C,$F700,$1000,$0000
        dc.w        $F84D,$F900,$1000,$0000,$FAAC,$FB00,$1000,$0000
        dc.w        $FCB2,$FD00,$1000,$0000,$FE9A,$FF00,$1000,$0000
        dc.w        $009A,$0106,$1000,$0000,$02DF,$0306,$1000,$0000
        dc.w        $0446,$0506,$1000,$0000,$0688,$0706,$1000,$0000
        dc.w        $0899,$0906,$1000,$0000,$0ADD,$0B06,$1000,$0000
        dc.w        $0CEE,$0D06,$1000,$0000,$0EFF,$0F06,$1000,$0000
        dc.w        $10CD,$1106,$1000,$0000,$1267,$1306,$1000,$0000
        dc.w        $1443,$1506,$1000,$0000,$1664,$1706,$1000,$0000
        dc.w        $1823,$1906,$1000,$0000,$1A6D,$1B06,$1000,$0000
        dc.w        $1C4F,$1D06,$1000,$0000,$1E5F,$1F06,$1000,$0000
        dc.w        $2055,$2106,$1000,$0000,$2267,$2306,$1000,$0000
        dc.w        $2445,$2506,$1000,$0000,$2623,$2706,$1000,$0000
        dc.w        $2834,$2906,$1000,$0000,$2AF0,$2B06,$1000,$0000
        dc.w        $2CBC,$2D06,$1000,$0000
SpriteEnd
        dc.w         $0000,$0000        ; Finalmente lo sprite riutilizzato ha termine


        SECTION        PLANEVUOTO,BSS_C
BITPLANE:
        ds.b        40*256

        end
```

In questo listato vediamo un classico effetto dell'Amiga.
Il cielo stellato e' realizzato mediante un unico sprite che viene riutilizzato
ben 127 volte, ogni volta ad una posizione verticale differente.
Una singola stella corrisponde ad un singolo utilizzo dello sprite.
Una stella e' alta solo una riga, e contiene un solo pixel colorato con il
colore 17 (cioe' con il primo "piano" a 1 e il secondo a 0).
Gli altri pixel sono tutti trasparenti,cioe' hanno entrambi i "piani" a 0.
Guardiamo come e' formata una singola stella, per esempio quella alla posizione
verticale $a0:

```
        dc.w    $A046,$A100,$1000,$0000
```

Notate che VSTART=$a0 , VSTOP=$a1, HSTART=$46. Una stella occupa in memoria
4 word, ovvero 8 bytes.
Come vedete VSTART e VSTOP differiscono di 1, il che indica che lo sprite e'
utilizzato per una sola riga. Le 2 word $1000 e $0000 sono i 2 "piani" che
contengono i dati sulla forma della prima e unica riga che forma la stella.
Dopo che e' servito per mostrare la stella alla posizione verticale $a0,
lo sprite viene usato per mostrare una stella alla posizione verticale $a2.
Guardiamo come sono disposti i dati di entrambe le stelle:

```
        dc.w    $A046,$A100,$1000,$0000,$A2A6,$A300,$1000,$0000
```

Come potete vedere, la stella alla posizione verticale $a2, e' fatta nello
stesso modo. Le uniche differenze sono nella posizione verticale (ovviamente)
e anche in quella orizzontale.

A questo punto vediamo come funziona la routine MuoviStelle.
La routine modifica la posizione delle stelle, una per volta, cominciando
dalla prima. Per modificare la posizione, viene aggiunto 1 al byte HSTART dello
sprite. Come ben saprete, in questo modo lo sprite viene mosso a destra di 2
pixel, perche' il bit basso di HSTART e' messo nel quarto byte di controllo.
Vedremo nel prossimo esempio come togliere questa limitazione. Poiche' come
abbiamo visto una stella occupa in memoria 8 bytes, ogni volta viene aggiunto
8 all'indirizzo dello sprite per puntare alla stella seguente.
Quando il puntatore raggiunge l'ultima stella, la routine termina.

Lezione7t2

```
; Lezione7t2.s        stelle lente

;        In questo listato facciamo un cielo stellato
;        con stelle che si muovono di un pixel per volta


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1         ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop


        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo gli sprite

        MOVE.L        #SPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviStelle        ; questa routine muove le stelle

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts
```

```
;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa semplice routine fa avanzare lo sprite che costituisce le stelle
; agendo su ogni HSTART (e sul byte VHBITS per lo scroll fluido di 1 pixel
; anziche' 2) tramite il loop STELLE_LOOP

MuoviStelle:
        lea        Sprite,A0                ; a0 punta allo sprite
STELLE_LOOP:
        CMPI.B       #$f0,1(A0)                ; la stella ha raggiunto il bordo
                                       ; destro dello schermo ?
        BNE.S        no_bordo                ; no, allora salta
        MOVE.B       #$30,1(A0)                ; si, rimetti la stella a sinistra
no_bordo:
        BCHG        #0,3(A0)                ; agisci sul byte VHBITS
        BEQ.S        pari
        ADDQ.B  #$1,1(A0)                ; sposta di 2 pixel lo sprite
pari:
        ADDQ.w       #8,A0                        ; prossima stella
        CMP.L        #SpriteEnd,A0                ; abbiamo raggiunto la fine?
        BLO.S        STELLE_LOOP                ; se no rifai il loop
        RTS                                ; fine routine


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop

        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$000        ; color1        ; colore 1 del bitplane, che
                                            ; in questo caso e' vuoto,
                                            ; per cui non compare.
```

```
; lo sprite usa un solo colore, il 17

        dc.w        $1A2,$ddd        ; color17, - COLOR1 dello sprite0 - bianco

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        -        -        -        -        -        -        -

; Come potete notare, questo sprite e' riutilizzato un bel po' di volte, per
; la precisione 127. E' composto di tante coppie di word di controllo seguite
; da un $1000,$0000, ossia la linea di dati che compone ogni singola stella.
; Le posizioni verticali vanno di 2 in 2, per cui c'e' una stella ogni 2
; linee verticali, ad una posizione orizzontale diversa per ogni stella.
; il primo utilizzo dello sprite e' $307a,$3100,$1000,$0000
; il secondo e' $3220,$3300,$1000,$0000
; E cosi' via. I vari VSTART vanno di 2 in 2, infatti sono $30,$32,$34,$36...
; Gli HSTART sono posizionati casualmente ($7a,$20,$c0,$50...)
; I Vstop sono chiaramente 1 linea dopo lo start ($31,$33,$35..) essendo le
; stelle alte 1 pixel.
; La routine agisce ogni fotogramma su tutti gli HSTART spostando avanti le
; stelle.

Sprite:
        dc.w    $307A,$3100,$1000,$0000,$3220,$3300,$1000,$0000
        dc.w    $34C0,$3500,$1000,$0000,$3650,$3700,$1000,$0000
        dc.w    $3842,$3900,$1000,$0000,$3A6D,$3B00,$1000,$0000
        dc.w    $3CA2,$3D00,$1000,$0000,$3E9C,$3F00,$1000,$0000
        dc.w    $40DA,$4100,$1000,$0000,$4243,$4300,$1000,$0000
        dc.w    $445A,$4500,$1000,$0000,$4615,$4700,$1000,$0000
        dc.w    $4845,$4900,$1000,$0000,$4A68,$4B00,$1000,$0000
        dc.w    $4CB8,$4D00,$1000,$0000,$4EB4,$4F00,$1000,$0000
        dc.w    $5082,$5100,$1000,$0000,$5292,$5300,$1000,$0000
        dc.w    $54D0,$5500,$1000,$0000,$56D3,$5700,$1000,$0000
        dc.w    $58F0,$5900,$1000,$0000,$5A6A,$5B00,$1000,$0000
        dc.w    $5CA5,$5D00,$1000,$0000,$5E46,$5F00,$1000,$0000
        dc.w    $606A,$6100,$1000,$0000,$62A0,$6300,$1000,$0000
        dc.w    $64D7,$6500,$1000,$0000,$667C,$6700,$1000,$0000
        dc.w    $68C4,$6900,$1000,$0000,$6AC0,$6B00,$1000,$0000
        dc.w    $6C4A,$6D00,$1000,$0000,$6EDA,$6F00,$1000,$0000
        dc.w    $70D7,$7100,$1000,$0000,$7243,$7300,$1000,$0000
        dc.w    $74A2,$7500,$1000,$0000,$7699,$7700,$1000,$0000
        dc.w    $7872,$7900,$1000,$0000,$7A77,$7B00,$1000,$0000
        dc.w    $7CC2,$7D00,$1000,$0000,$7E56,$7F00,$1000,$0000
        dc.w    $805A,$8100,$1000,$0000,$82CC,$8300,$1000,$0000
        dc.w    $848F,$8500,$1000,$0000,$8688,$8700,$1000,$0000
        dc.w    $88B9,$8900,$1000,$0000,$8AAF,$8B00,$1000,$0000
        dc.w    $8C48,$8D00,$1000,$0000,$8E68,$8F00,$1000,$0000
        dc.w    $90DF,$9100,$1000,$0000,$924F,$9300,$1000,$0000
        dc.w    $9424,$9500,$1000,$0000,$96D7,$9700,$1000,$0000
        dc.w    $9859,$9900,$1000,$0000,$9A4F,$9B00,$1000,$0000
        dc.w    $9C4A,$9D00,$1000,$0000,$9E5C,$9F00,$1000,$0000
        dc.w    $A046,$A100,$1000,$0000,$A2A6,$A300,$1000,$0000
        dc.w    $A423,$A500,$1000,$0000,$A6FA,$A700,$1000,$0000
        dc.w    $A86C,$A900,$1000,$0000,$AA44,$AB00,$1000,$0000
        dc.w    $AC88,$AD00,$1000,$0000,$AE9A,$AF00,$1000,$0000
        dc.w    $B06C,$B100,$1000,$0000,$B2D4,$B300,$1000,$0000
        dc.w    $B42A,$B500,$1000,$0000,$B636,$B700,$1000,$0000
        dc.w    $B875,$B900,$1000,$0000,$BA89,$BB00,$1000,$0000
        dc.w    $BC45,$BD00,$1000,$0000,$BE24,$BF00,$1000,$0000
        dc.w    $C0A3,$C100,$1000,$0000,$C29D,$C300,$1000,$0000
        dc.w    $C43F,$C500,$1000,$0000,$C634,$C700,$1000,$0000
        dc.w    $C87C,$C900,$1000,$0000,$CA1D,$CB00,$1000,$0000
```

```
        dc.w    $CC6B,$CD00,$1000,$0000,$CEAC,$CF00,$1000,$0000
        dc.w    $D0CF,$D100,$1000,$0000,$D2FF,$D300,$1000,$0000
        dc.w    $D4A5,$D500,$1000,$0000,$D6D6,$D700,$1000,$0000
        dc.w    $D8EF,$D900,$1000,$0000,$DAE1,$DB00,$1000,$0000
        dc.w    $DCD9,$DD00,$1000,$0000,$DEA6,$DF00,$1000,$0000
        dc.w    $E055,$E100,$1000,$0000,$E237,$E300,$1000,$0000
        dc.w    $E47D,$E500,$1000,$0000,$E62E,$E700,$1000,$0000
        dc.w    $E8AF,$E900,$1000,$0000,$EA46,$EB00,$1000,$0000
        dc.w        $EC65,$ED00,$1000,$0000,$EE87,$EF00,$1000,$0000
        dc.w        $F0D4,$F100,$1000,$0000,$F2F5,$F300,$1000,$0000
        dc.w        $F4FA,$F500,$1000,$0000,$F62C,$F700,$1000,$0000
        dc.w        $F84D,$F900,$1000,$0000,$FAAC,$FB00,$1000,$0000
        dc.w        $FCB2,$FD00,$1000,$0000,$FE9A,$FF00,$1000,$0000
        dc.w        $009A,$0106,$1000,$0000,$02DF,$0306,$1000,$0000
        dc.w        $0446,$0506,$1000,$0000,$0688,$0706,$1000,$0000
        dc.w        $0899,$0906,$1000,$0000,$0ADD,$0B06,$1000,$0000
        dc.w        $0CEE,$0D06,$1000,$0000,$0EFF,$0F06,$1000,$0000
        dc.w        $10CD,$1106,$1000,$0000,$1267,$1306,$1000,$0000
        dc.w        $1443,$1506,$1000,$0000,$1664,$1706,$1000,$0000
        dc.w        $1823,$1906,$1000,$0000,$1A6D,$1B06,$1000,$0000
        dc.w        $1C4F,$1D06,$1000,$0000,$1E5F,$1F06,$1000,$0000
        dc.w        $2055,$2106,$1000,$0000,$2267,$2306,$1000,$0000
        dc.w        $2445,$2506,$1000,$0000,$2623,$2706,$1000,$0000
        dc.w        $2834,$2906,$1000,$0000,$2AF0,$2B06,$1000,$0000
        dc.w        $2CBC,$2D06,$1000,$0000
SpriteEnd
        dc.w         $0000,$0000        ; fine sprite


        SECTION        PLANEVUOTO,BSS_C
BITPLANE:
        ds.b        40*256

        end
```

In questo listato facciamo muovere le stelle a passi di un pixel.
Il programma e' identico all'esempio precedente, la sola cosa che cambia e'
che dobbiamo aumentare di 1 pixel invece che di 2 le posizioni orizzontali
delle stelle quando le mouviamo. Il problema e' che come sapete il bit basso
della posizione orizzontale di uno sprite e' scritto separatamente dagli
altri, quindi non possiamo spostare di 1 pixel uno sprite con una semplice
ADD. Una possibile soluzione sarebbe quella di tenere memorizzate a parte le
posizioni orizzontali in forma di word, cioe' con tutti i bit insieme,
aggiungere 1 alle word e poi scriverle nel byte HSTART e nel bit basso
mediante la stessa procedura che abbiamo usato nella routine UniMuoviSprite.
Questa soluzione ci costringerebbe pero' a memorizzare ben 127 word, una per
ogni posizione orizzontale. Nella routine abbiamo usato un metodo migliore.

```
        BCHG        #0,3(A0)                ; agisci sul byte VHBITS
        BEQ.S        pari
        ADDQ.B  #$1,1(A0)                ; sposta di 2 pixel lo sprite
pari:
```

vediamo passo per passo cosa succede

```
        BCHG        #0,3(A0)                ; agisci sul byte VHBITS
```

questa istruzione funziona esattamente come il BSET o il BCLR, solo che
mentre BSET pone sempre a 1 il bit indicato e BCLR lo pone sempre a 0,
BCHG lo inverte: se prima era 0 lo trasforma in 1 e viceversa. In questo caso
il bit in questione e' proprio il bit basso della posizione della stella.
Inoltre BCHG PRIMA DI INVERTIRE IL BIT, controlla se esso e' a 0, esattamente

```
come fa il BTST. Quindi l'istruzione

        BEQ.S        pari

effettua il salto se il bit basso di HSTART ERA A 0 PRIMA DI ESEGUIRE IL BCHG.

L'istruzione seguente sposta di 2 pixel lo sprite, se la posizione era dispari

        ADDQ.B  #$1,1(A0)  ; sposta di 2 pixel lo sprite.

Queste 3 istruzioni risolvono il nostro problema. Infatti si possono
verificare 2 casi:

Se il bit basso ERA a 0, il BCHG lo porta a 1, il che equivale ad aggiungere
1 alla posizione orizzontale. Poiche' in questo caso il BEQ effettua il
salto, il byte HSTART non viene modificato. In pratica se per esempio avevamo
byte HSTART=%0110010 e bit basso = 0 (che messi insieme formavano per la
posizione orizzontale il numero %01100100=100), dopo aver eseguito la routine
ci ritroviamo con HSTART uguale e il bit basso=1, ovvero
byte HSTART=%0110010 e bit basso = 1 (che messi insieme formano per la
posizione orizzontale il numero %01100101=101).
Come volevamo abbiamo aumentato di 1 la posizione orizzontale.

Se invece il bit basso ERA a 1 la  BCHG lo porta a 0, il che equivale a
sottrarre 1 alla posizione orizzontale. In questo caso pero' il BEQ
NON effettua il salto, e pertanto viene aggiunto 1 al il byte HSTART il che
come sapete equivale ad aggiungere 2 alla posizione orizzontale. Combinando
insieme la sottrazione di 1 fatta dal BCHG con la somma di 2 fatta dalla
ADD, abbiamo aggiunto 1 alla posizione orizzontale.

In pratica se per esempio prima di eseguire la routine avevamo
byte HSTART=%0110010 e bit basso = 1 (che messi insieme formavano per la
posizione orizzontale il numero %01100101=101), dopo aver eseguito la routine
ci ritroviamo con HSTART aumentato di 1 e il bit basso=0, ovvero
byte HSTART=%0110011 e bit basso = 0 che messi insieme formano per la
posizione orizzontale il numero %01100110=102).
Come volevamo anche in questo caso abbiamo aumentato di 1 la posizione
orizzontale dello sprite.
```

Lezione7t3

```
; Lezione7t3.s        stelle a diverse velocita'

;        In questo listato facciamo un cielo stellato
;        con stelle che hanno diverse velocita'

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop


        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
```

```
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo gli sprite

        MOVE.L        #SPRITE,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviStelle        ; questa routine muove le stelle

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; La routine che muove le stelle e' un poco piu' complessa in questo esempio,
; ma basta seguirla istruzione per istruzione e non e' difficile comprenderla.
; Per ogni STELLE_LOOP la routine muove 3 stelle, per la precisione 1 veloce,
; 1 lenta e 1 velocissima.

MuoviStelle:
        lea        Sprite,A0                ; a0 punta allo sprite
STELLE_LOOP:

; stelle veloci (2 pixel per fotogramma)

        CMPI.B        #$f0,1(A0)                ; la stella ha raggiunto il bordo
```

```
                                    ; destro dello schermo ?
        BNE.S        no_bordo1              ; no, allora salta
        MOVE.B        #$30,1(A0)              ; si, rimetti la stella a sinistra
no_bordo1:
        ADDQ.B  #1,1(A0)               ; sposta di 2 pixel lo sprite
        ADDQ.w        #8,A0                    ; prossima stella

; stelle lente (1 pixel per fotogramma)

         CMPI.B        #$f0,1(A0)               ; la stella ha raggiunto il bordo
                                    ; destro dello schermo ?
        BNE.S        no_bordo2              ; no, allora salta
        MOVE.B        #$30,1(A0)              ; si, rimetti la stella a sinistra
no_bordo2:
        BCHG        #0,3(A0)
        BEQ.S        pari
        ADDQ.B  #1,1(A0)               ; sposta di 2 pixel lo sprite
pari:
        ADDQ.w        #8,A0                    ; prossima stella

; stelle velocissime (4 pixel per fotogramma)

        CMPI.B        #$f0,1(A0)               ; la stella ha raggiunto il bordo
                                    ; destro dello schermo ?
        BNE.S        no_bordo3              ; no, allora salta
        MOVE.B        #$30,1(A0)              ; si, rimetti la stella a sinistra
no_bordo3:
        ADDQ.B  #2,1(A0)               ; sposta di 4 pixel lo sprite
        ADDQ.w        #8,A0                    ; prossima stella

        CMP.L        #SpriteEnd,A0             ; abbiamo raggiunto la fine?
        BLO.S        STELLE_LOOP              ; se no rifai il loop
        RTS                                ; fine routine


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop

        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$000        ; color1        ; colore 1 del bitplane, che
```

```
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.
; lo sprite usa un solo colore, il 17

        dc.w        $1A2,$ddd        ; color17, - COLOR1 dello sprite0 - bianco

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        -        -        -        -        -        -        -

; Come potete notare, questo sprite e' riutilizzato un bel po' di volte, per
; la precisione 127. E' composto di tante coppie di word di controllo seguite
; da un $1000,$0000, ossia la linea di dati che compone ogni singola stella.
; Le posizioni verticali vanno di 2 in 2, per cui c'e' una stella ogni 2
; linee verticali, ad una posizione orizzontale diversa per ogni stella.
; il primo utilizzo dello sprite e' $307a,$3100,$1000,$0000
; il secondo e' $3220,$3300,$1000,$0000
; E cosi' via. I vari VSTART vanno di 2 in 2, infatti sono $30,$32,$34,$36...
; Gli HSTART sono posizionati casualmente ($7a,$20,$c0,$50...)
; I Vstop sono chiaramente 1 linea dopo lo start ($31,$33,$35..) essendo le
; stelle alte 1 pixel.
; La routine agisce ogni fotogramma su tutti gli HSTART spostando avanti le
; stelle.

Sprite:
        dc.w    $307A,$3100,$1000,$0000,$3220,$3300,$1000,$0000
        dc.w    $34C0,$3500,$1000,$0000,$3650,$3700,$1000,$0000
        dc.w    $3842,$3900,$1000,$0000,$3A6D,$3B00,$1000,$0000
        dc.w    $3CA2,$3D00,$1000,$0000,$3E9C,$3F00,$1000,$0000
        dc.w    $40DA,$4100,$1000,$0000,$4243,$4300,$1000,$0000
        dc.w    $445A,$4500,$1000,$0000,$4615,$4700,$1000,$0000
        dc.w    $4845,$4900,$1000,$0000,$4A68,$4B00,$1000,$0000
        dc.w    $4CB8,$4D00,$1000,$0000,$4EB4,$4F00,$1000,$0000
        dc.w    $5082,$5100,$1000,$0000,$5292,$5300,$1000,$0000
        dc.w    $54D0,$5500,$1000,$0000,$56D3,$5700,$1000,$0000
        dc.w    $58F0,$5900,$1000,$0000,$5A6A,$5B00,$1000,$0000
        dc.w    $5CA5,$5D00,$1000,$0000,$5E46,$5F00,$1000,$0000
        dc.w    $606A,$6100,$1000,$0000,$62A0,$6300,$1000,$0000
        dc.w    $64D7,$6500,$1000,$0000,$667C,$6700,$1000,$0000
        dc.w    $68C4,$6900,$1000,$0000,$6AC0,$6B00,$1000,$0000
        dc.w    $6C4A,$6D00,$1000,$0000,$6EDA,$6F00,$1000,$0000
        dc.w    $70D7,$7100,$1000,$0000,$7243,$7300,$1000,$0000
        dc.w    $74A2,$7500,$1000,$0000,$7699,$7700,$1000,$0000
        dc.w    $7872,$7900,$1000,$0000,$7A77,$7B00,$1000,$0000
        dc.w    $7CC2,$7D00,$1000,$0000,$7E56,$7F00,$1000,$0000
        dc.w    $805A,$8100,$1000,$0000,$82CC,$8300,$1000,$0000
        dc.w    $848F,$8500,$1000,$0000,$8688,$8700,$1000,$0000
        dc.w    $88B9,$8900,$1000,$0000,$8AAF,$8B00,$1000,$0000
        dc.w    $8C48,$8D00,$1000,$0000,$8E68,$8F00,$1000,$0000
        dc.w    $90DF,$9100,$1000,$0000,$924F,$9300,$1000,$0000
        dc.w    $9424,$9500,$1000,$0000,$96D7,$9700,$1000,$0000
        dc.w    $9859,$9900,$1000,$0000,$9A4F,$9B00,$1000,$0000
        dc.w    $9C4A,$9D00,$1000,$0000,$9E5C,$9F00,$1000,$0000
        dc.w    $A046,$A100,$1000,$0000,$A2A6,$A300,$1000,$0000
        dc.w    $A423,$A500,$1000,$0000,$A6FA,$A700,$1000,$0000
        dc.w    $A86C,$A900,$1000,$0000,$AA44,$AB00,$1000,$0000
        dc.w    $AC88,$AD00,$1000,$0000,$AE9A,$AF00,$1000,$0000
        dc.w    $B06C,$B100,$1000,$0000,$B2D4,$B300,$1000,$0000
        dc.w    $B42A,$B500,$1000,$0000,$B636,$B700,$1000,$0000
        dc.w    $B875,$B900,$1000,$0000,$BA89,$BB00,$1000,$0000
        dc.w    $BC45,$BD00,$1000,$0000,$BE24,$BF00,$1000,$0000
        dc.w    $C0A3,$C100,$1000,$0000,$C29D,$C300,$1000,$0000
```

```
        dc.w    $C43F,$C500,$1000,$0000,$C634,$C700,$1000,$0000
        dc.w    $C87C,$C900,$1000,$0000,$CA1D,$CB00,$1000,$0000
        dc.w    $CC6B,$CD00,$1000,$0000,$CEAC,$CF00,$1000,$0000
        dc.w    $D0CF,$D100,$1000,$0000,$D2FF,$D300,$1000,$0000
        dc.w    $D4A5,$D500,$1000,$0000,$D6D6,$D700,$1000,$0000
        dc.w    $D8EF,$D900,$1000,$0000,$DAE1,$DB00,$1000,$0000
        dc.w    $DCD9,$DD00,$1000,$0000,$DEA6,$DF00,$1000,$0000
        dc.w    $E055,$E100,$1000,$0000,$E237,$E300,$1000,$0000
        dc.w    $E47D,$E500,$1000,$0000,$E62E,$E700,$1000,$0000
        dc.w    $E8AF,$E900,$1000,$0000,$EA46,$EB00,$1000,$0000
        dc.w        $EC65,$ED00,$1000,$0000,$EE87,$EF00,$1000,$0000
        dc.w        $F0D4,$F100,$1000,$0000,$F2F5,$F300,$1000,$0000
        dc.w        $F4FA,$F500,$1000,$0000,$F62C,$F700,$1000,$0000
        dc.w        $F84D,$F900,$1000,$0000,$FAAC,$FB00,$1000,$0000
        dc.w        $FCB2,$FD00,$1000,$0000,$FE9A,$FF00,$1000,$0000
        dc.w        $009A,$0106,$1000,$0000,$02DF,$0306,$1000,$0000
        dc.w        $0446,$0506,$1000,$0000,$0688,$0706,$1000,$0000
        dc.w        $0899,$0906,$1000,$0000,$0ADD,$0B06,$1000,$0000
        dc.w        $0CEE,$0D06,$1000,$0000,$0EFF,$0F06,$1000,$0000
        dc.w        $10CD,$1106,$1000,$0000,$1267,$1306,$1000,$0000
        dc.w        $1443,$1506,$1000,$0000,$1664,$1706,$1000,$0000
        dc.w        $1823,$1906,$1000,$0000,$1A6D,$1B06,$1000,$0000
        dc.w        $1C4F,$1D06,$1000,$0000,$1E5F,$1F06,$1000,$0000
        dc.w        $2055,$2106,$1000,$0000,$2267,$2306,$1000,$0000
        dc.w        $2445,$2506,$1000,$0000,$2623,$2706,$1000,$0000
        dc.w        $2834,$2906,$1000,$0000,$2AF0,$2B06,$1000,$0000
SpriteEnd:
        dc.w         $0000,$0000        ; Fine dello sprite


        SECTION        PLANEVUOTO,BSS_C
BITPLANE:
        ds.b        40*256

        end
```

In questo listato facciamo muovere le stelle a velocita‘ diversa, in modo da creare un effetto migliore. Diamo alle stelle tre velocita‘ diverse.
La routine MuoviStelle sposta tre stelle per ogni ciclo. La prima stella viene spostata di 2 pixel come abbiamo visto nel primo esempio, la seconda stella di 1 solo pixel, mentre la terza viene spostata di 4 pixel, aggiungendo 2 al byte HSTART. La routine controlla se abbiamo raggiunto l’ultima stella solo dopo averne spostate 3.

Lezione7t4

```
; Lezione7t4.s        palline

;        In questo listato facciamo una serie di palline in movimento
;        riutilizzando i 4 sprite "attached" 11 volte ciascuno, per un totale
;        di 44 palline.
;        Ognuno degli sprite attacched viene riutilizzato per formare uno
;        "strato" di stelle con la medesima velocita', per cui ci sono
;        4 diverse velocita' di scorrimento.
;        Ad esempio le stelle piu' piccole e lente, che sembrano le piu'
;        lontane, sono tutte fatte col riutilizzo dello sprite attacched 4,
;        ossia formato dagli sprite 6 e 7 attaccati.


        SECTION        CiriCop,CODE
```

```
Inizio:
        move.l      4.w,a6              ; Execbase
        jsr        -$78(a6)         ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)         ; OpenLibrary
        move.l      d0,GfxBase
        move.l      d0,a6
        move.l      $26(a6),OldCop


        MOVE.L      #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

;        Puntiamo gli sprite

        MOVE.L      #SPRITE0,d0              ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

;        Puntiamo tutti gli 8 sprite, dato che li utiliziamo tutti per fare
;        4 sprite attacched, i quali formano i 4 "livelli" di stelle a
;        diversa velocita'

        MOVE.L      #SPRITE1,d0
        addq.w      #8,a1
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        MOVE.L      #SPRITE2,d0
        addq.w      #8,a1
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        MOVE.L      #SPRITE3,d0
        addq.w      #8,a1
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        MOVE.L      #SPRITE4,d0
        addq.w      #8,a1
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        MOVE.L      #SPRITE5,d0
        addq.w      #8,a1
        move.w      d0,6(a1)
        swap        d0
        move.w      d0,2(a1)

        MOVE.L      #SPRITE6,d0
        addq.w      #8,a1
        move.w      d0,6(a1)
```

```
        swap        d0
        move.w       d0,2(a1)

        MOVE.L       #SPRITE7,d0
        addq.w       #8,a1
        move.w       d0,6(a1)
        swap        d0
        move.w       d0,2(a1)

        move.l       #COPPERLIST,$dff080        ; nostra COP
        move.w       d0,$dff088                ; START COP
        move.w       #0,$dff1fc                ; NO AGA!
        move.w       #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b       #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviSprites_01        ; questa routine muove gli sprite 0 e 1
                     ; attacched, cioe' le palline grandi con
                     ; la massima velocita': 8 pixel alla volta

        bsr.s        MuoviSprites_23        ; questa routine muove gli sprite 2 e 3
                     ; attacched, cioe' le palline grandi con
                     ; una velocita' di 6 pixel alla volta

        bsr.w        MuoviSprites_45        ; questa routine muove gli sprite 4 e 5
                     ; attacched, cioe' le palline di media
                     ; grandezza e media velocita' (4 pixel a volta)

        bsr.w        MuoviSprites_67        ; questa routine muove gli sprite 6 e 7
                     ; attacched, cioe' le palline piu' lente
                     ; e piccole (spostate di 2 pixel alla volta)

Aspetta:
        cmpi.b       #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr         -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr         -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine muove gli sprite 0 e 1 che sono attaccati, pertanto devono
```

```
; avere le stesse coordinate.

MuoviSprites_01:
        lea        Sprite0,a0        ; indirizzo sprite 0
        lea        Sprite1,a1        ; indirizzo sprite 1
        moveq        #11-1,d7        ; numero di utilizzi dello sprite
loop01:
        addq.b        #4,1(a0)        ; sposta di 8 pixel a destra lo sprite 0
                                ; agendo sul suo HSTART
        addq.b        #4,1(a1)        ; sposta di 8 pixel a destra lo sprite 1
                                ; agendo sul suo HSTART
        lea        68(a0),a0        ; coordinate prossimo riuso dello sprite 0
        lea        68(a1),a1        ; coordinate prossimo riuso dello sprite 1
        dbra        d7,loop01       ; loop
        rts

; Questa routine muove gli sprite 2 e 3 che sono attaccati, pertanto devono
; avere le stesse coordinate.

MuoviSprites_23:
        lea        Sprite2,a0        ; indirizzo sprite 2
        lea        Sprite3,a1        ; indirizzo sprite 3
        moveq        #11-1,d7        ; numero di utilizzi dello sprite
loop23:
        addq.b        #3,1(a0)        ; sposta di 6 pixel a destra lo sprite 2
                                ; agendo sul suo HSTART
        addq.b        #3,1(a1)        ; sposta di 6 pixel a destra lo sprite 3
                                ; agendo sul suo HSTART
        lea        68(a0),a0        ; coordinate prossimo riuso dello sprite 2
        lea        68(a1),a1        ; coordinate prossimo riuso dello sprite 3
        dbra        d7,loop23       ; loop
        rts

; Questa routine muove gli sprite 4 e 5 che sono attaccati, pertanto devono
; avere le stesse coordinate.

MuoviSprites_45:
        lea        Sprite4,a0        ; indirizzo sprite 4
        lea        Sprite5,a1        ; indirizzo sprite 5
        moveq        #11-1,d7        ; numero di utilizzi dello sprite
loop45:
        addq.b        #2,1(a0)        ; sposta di 4 pixel a destra lo sprite 4
        addq.b        #2,1(a1)        ; sposta di 4 pixel a destra lo sprite 5
        lea        68(a0),a0        ; coordinate prossimo riuso dello sprite 4
        lea        68(a1),a1        ; coordinate prossimo riuso dello sprite 5
        dbra        d7,loop45       ; loop
        rts

; Questa routine muove gli sprite 4 e 5 che sono attaccati, pertanto devono
; avere le stesse coordinate.

MuoviSprites_67:
        lea        Sprite6,a0        ; indirizzo sprite 6
        lea        Sprite7,a1        ; indirizzo sprite 7
        moveq        #11-1,d7        ; numero di utilizzi dello sprite
loop67:
        addq.b        #1,1(a0)        ; sposta di 2 pixel a destra lo sprite 6
        addq.b        #1,1(a1)        ; sposta di 2 pixel a destra lo sprite 7
        lea        68(a0),a0        ; coordinate prossimo riuso dello sprite 6
        lea        68(a1),a1        ; coordinate prossimo riuso dello sprite 7
        dbra        d7,loop67       ; loop
        rts
```

```
        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop

        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$000        ; color1        ; colore 1 del bitplane, che
                                                     ; in questo caso e' vuoto,
                                                     ; per cui non compare.

        dc.w        $1a0,$000,$1a2,$fff        ; palette degli sprites
        dc.w        $1a4,$f00,$1a6,$b00
        dc.w        $1a8,$600,$1aa,$F40
        dc.w        $1ac,$F80,$1ae,$Fa0
        dc.w        $1b0,$Ff0,$1b2,$00f
        dc.w        $1b4,$04f,$1b6,$08f
        dc.w        $1b8,$0ff,$1ba,$0f0
        dc.w        $1bc,$283,$1be,$f0f


        dc.w        $FFFF,$FFFE        ; Fine della copperlist

; Qui ci sono gli sprite. Ogni sprite e‘ riutilizzato 11 volte.
; Gli sprite dispari hanno il bit ATTACHED settato per formare sprite
; a 16 colori. Come potete notare, le "palline" sono tutte uguali.

Sprite0:
        dc.w    $38D0,$4800        ; words di controllo
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318 ; pallina
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w        $4943,$5900        ; words di controllo
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318 ; pallina
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w        $6087,$7000
```

```
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w    $71af,$8100
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w    $8213,$9200
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w    $93D0,$a300
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w        $a443,$b400
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w        $b587,$c500
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w    $c6af,$d600
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w    $d713,$e700
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

        dc.w    $e8b9,$f800
        dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
        dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000
        dc.w        0,0         ; fine sprite0

Sprite1:
        dc.w    $38D0,$4880         ; words di controllo
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000 ;pallina
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $4943,$5980         ; words di controllo
```

```
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000 ; pallina
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $6087,$7080
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $71af,$8180
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $8213,$9280
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $93D0,$a380
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $a443,$b480
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $b587,$c580
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $c6af,$d680
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $d713,$e780
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $e8b9,$f880
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000
        dc.w        0,0 ; fine sprite 1

Sprite2:
        dc.w    $44D0,$5400         ; words di controllo
```

```
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318 ; pallina
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w        $5543,$6500        ; words di controllo
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318 ; pallina
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w        $6687,$7600
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w    $77af,$8700
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w    $8813,$9800
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w    $99D0,$a900
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w        $aa43,$ba00
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w        $bb87,$cb00
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w    $ccaf,$dc00
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w    $dd13,$ed00
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000

dc.w    $ee5c,$fe00
dc.w        $0000,$0000,$0200,$0200,$0db0,$0d80,$1520,$1318
dc.w        $2e30,$3208,$3e70,$260c,$3464,$2c1c,$70e0,$7018
```

```
        dc.w        $20c8,$2038,$01c0,$0030,$0390,$0070,$0720,$00e0
        dc.w        $0e40,$01c0,$0000,$0700,$0000,$0000,$0000,$0000
        dc.w        0,0         ; fine sprite 2

Sprite3:
        dc.w    $44D0,$5480         ; words di controllo
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000 ; pallina
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $5543,$6580         ; words di controllo
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000 ; pallina
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $6687,$7680
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $77af,$8780
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $8813,$9880
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $99D0,$a980
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $aa43,$ba80
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w        $bb87,$cb80
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $ccaf,$dc80
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $dd13,$ed80
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
```

```
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000

        dc.w    $ee5c,$fe80
        dc.w        $07c0,$0000,$1df8,$0000,$3278,$0000,$68fc,$0000
        dc.w        $41fc,$0000,$c1fe,$0000,$c3fe,$0000,$8ffa,$0004
        dc.w        $dffa,$0004,$fff2,$000c,$7ff4,$0008,$7fe4,$0018
        dc.w        $3fc8,$0030,$1f30,$00c0,$07c0,$0000,$0000,$0000
        dc.w        0,0         ; fine sprite 3

Sprite4:
        dc.w    $3877,$4800         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000 ; pallina
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w    $49D0,$5900         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000 ; pallina
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $6043,$7000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $7187,$8100
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w    $82af,$9200
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w    $9313,$a300
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w    $a4D0,$b400
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $b543,$c500
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $c687,$d600
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
```

```
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w    $d7af,$e700
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w    $e813,$f800
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0540,$0700,$0e60,$0980,$3cc0,$3220,$1a90,$1670
        dc.w        $0490,$1c70,$19a0,$1860,$0320,$00e0,$0640,$01c0
        dc.w        $0080,$0380,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        0,0         ; fine sprite 4

Sprite5:
        dc.w    $3877,$4880         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000 ; pallina
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w    $49D0,$5980         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000 ; pallina
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w        $6043,$7080
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w        $7187,$8180
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w    $82af,$9280
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w    $9313,$a380
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w    $a4D0,$b480
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w        $b543,$c580
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
```

```
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w        $c687,$d680
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w    $d7af,$e780
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000

        dc.w    $e813,$f880
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$07c0,$0000
        dc.w        $08e0,$0000,$1070,$0000,$01f8,$0000,$21f8,$0000
        dc.w        $23f8,$0000,$27e8,$0010,$3fe8,$0010,$1fd0,$0020
        dc.w        $0fa0,$0040,$07c0,$0000,$0000,$0000,$0000,$0000
        dc.w        0,0         ; fine sprite 5

Sprite6:
        dc.w        $4040,$5000         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000 ; pallina
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $5188,$6100         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000 ; pallina
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $6206,$7200
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $73dd,$8300
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $8469,$9400
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $95e4,$a500
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $a62c,$b600
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
```

```
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $b799,$c700
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $c8d0,$d800
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $d955,$e900
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $eab4,$fa00
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$03a0,$0280,$03e0,$00a0,$0340,$0320
        dc.w        $0180,$0140,$0340,$00c0,$0000,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        0,0

Sprite7:
        dc.w        $4040,$5080         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000 ; pallina
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $5188,$6180         ; words di controllo
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000 ; pallina
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $6206,$7280
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $73dd,$8380
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $8469,$9480
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $95e4,$a580
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
```

```
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $a62c,$b680
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $b799,$c780
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $c8d0,$d880
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $d955,$e980
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000

        dc.w        $eab4,$fa80
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        $01c0,$0000,$0060,$0000,$0470,$0000,$04f0,$0000
        dc.w        $06d0,$0020,$03e0,$0000,$01c0,$0000,$0000,$0000
        dc.w        $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
        dc.w        0,0         ; fine sprite 7


        SECTION         PLANEVUOTO,BSS_C
BITPLANE:
        ds.b        40*256

        end
```

In questo listato vediamo un miglioramento dell'effetto delle stelle.
Questa volta invece di stelle fatte da un puntino, muoviamo delle palline
colorate. Usiamo sempre degli sprite, ma a 16 colori, perche' ogni sferetta
e' costituita da una coppia di sprite attaccati. Inoltre non utiliziamo
una sola coppia di sprite attaccati (le stelle erano fatte con un solo sprite
riutilizzato), ma tutte e 4 le coppie disponibili, il che ci consente di
avere piu' sprite che viaggiano sulla stessa riga e che si sovrappongono.
Ogni coppia di sprite e' riutilizzata 11 volte per un totale di 44 palline
sullo schermo.

Utiliziamo per ogni coppia di sprite una routine di movimento separata.
Le quattro routine comunque si differenziano solo per la diversa velocita'
con cui muovono le palline. Palline create da una stessa coppia di sprite
hanno la stessa velocita', mentre palline create da coppie diverse hanno
velocita' diverse.

Per il resto non ci sono differenze con i listati delle stelle.

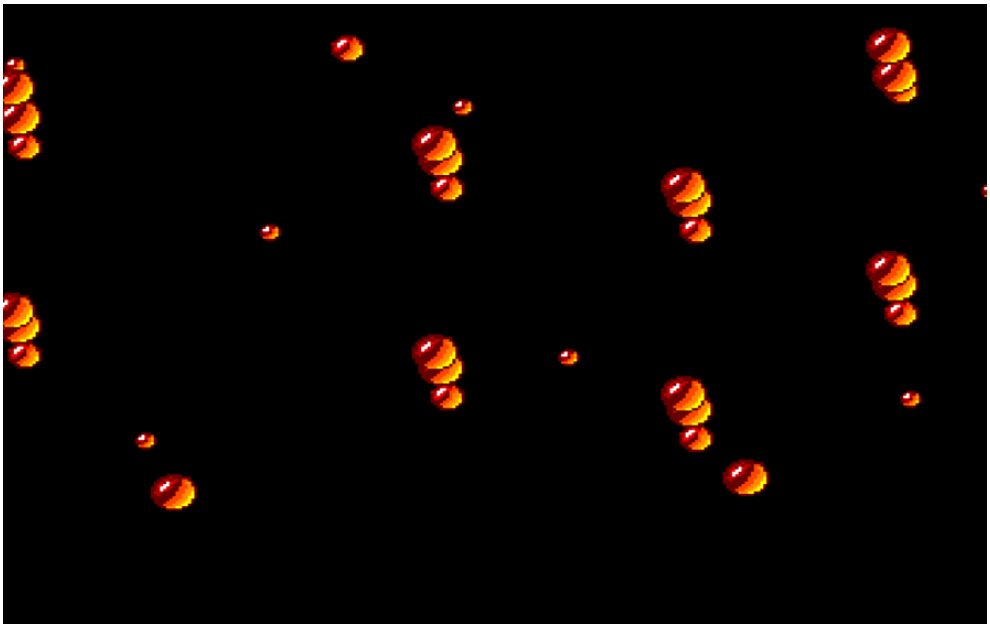


Figura 23.7: Lezione 7t4

23.19 Lezione7u

```
; lezione7u.s  - ESEMPIO Double Playfield
; Questo e‘ un semplice esempio del dual playfield mode.
; Sono visualizzati i due playfield. Premendo il bottone destro si cambia
; la priorita‘ dei due playfield. Sinistro per uscire
; Fate attenzione alla copperlist, perche’ le principali differenze del modo
; Dual Playfield rispetto al modo normale e’ nei BPLPOINTERS e nei colori.


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)         ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo le PIC

        MOVE.L        #PIC1,d0        ; puntiamo il playfield 1
        LEA         BPLPOINTERS1,A1
        MOVEQ         #3-1,D1
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
```

```
        ADD.L       #40*256,d0
        addq.w       #8,a1
        dbra       d1,POINTBP

        MOVE.L       #PIC2,d0        ; puntiamo il playfield 2
        LEA       BPLPOINTERS2,A1
        MOVEQ       #3-1,D1
POINTBP2:
        move.w       d0,6(a1)
        swap       d0
        move.w       d0,2(a1)
        swap       d0
        ADD.L       #40*256,d0
        addq.w       #8,a1
        dbra       d1,POINTBP2


        move.l       #COPPERLIST,$dff080        ; nostra COP
        move.w       d0,$dff088                ; START COP
        move.w       #0,$dff1fc                ; NO AGA!
        move.w       #$c00,$dff106                ; NO AGA!

mouse1:
        btst       #2,$dff016        ; mouse premuto?
        bne.s       mouse2

        bchg.b       #6,BPLCON2        ; scambiamo la priorita' agendo sul bit 6
                                ; del $dff104

mouse2:
        btst       #6,$bfe001        ; mouse premuto?
        bne.s       mouse1

        move.l       OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w       d0,$dff088                ; facciamo partire la vecchia cop

        move.l       4.w,a6
        jsr       -$7e(a6)        ; Enable
        move.l       gfxbase(PC),a1
        jsr       -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b       "graphics.library",0,0

GfxBase:
        dc.l       0

OldCop:
        dc.l       0


        SECTION       GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w       $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w       $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w       $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w       $13e,0
```

```
        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1

        dc.w        $104               ; BplCon2
        dc.b        0
BPLCON2:
        dc.b        0               ; priorita' fra playfields: bit 6

        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0110011000000000        ; bit 10 acceso = dual playfield
                                    ; uso 6 planes = 8 colori per playfield

BPLPOINTERS1:
        dc.w $e0,0,$e2,0        ;primo bitplane playfield 1 (BPLPT1)
        dc.w $e8,0,$ea,0        ;secondo bitplane playfield 1 (BPLPT3)
        dc.w $f0,0,$f2,0        ;terzo bitplane playfield 1 (BPLPT5)


BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ;primo bitplane playfield 2 (BPLPT2)
        dc.w $ec,0,$ee,0        ;secondo bitplane playfield 2 (BPLPT4)
        dc.w $f4,0,$f6,0        ;terzo bitplane playfield 2 (BPLPT6)

        dc.w        $180,$0f0        ; palette playfield 1
        dc.w        $182,$005        ; colori da 0 a 7
        dc.w        $184,$a40
        dc.w        $186,$f80
        dc.w        $188,$f00
        dc.w        $18a,$0f0
        dc.w        $18c,$00f
        dc.w        $18e,$080

                                ; palette playfield 2
        dc.w        $192,$367        ; colori da 9 a 15
        dc.w        $194,$0cc         ; il colore 8 e' trasparente, non va settato
        dc.w        $196,$a0a
        dc.w        $198,$242
        dc.w        $19a,$282
        dc.w        $19c,$861
        dc.w        $19e,$ff0

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Ecco le figure dei 2 playfields

PIC1:        incbin        "dual1.raw"
PIC2:        incbin        "dual2.raw"

        end
```

Questo e' un semplice esempio di uso del modo dual-playfield. Per visualizzare una schermata dual playfield si eseguono, piu' o meno, le stesse operazioni. Bisogna solo tenere presente che nel puntare i bit-planes, i dispari puntano ad un playfield e i pari all'altro. Di solito quindi si usano

```
2 routine separate, e anche nella copperlist i bit-planes dispari sono
separati dai pari. Per quanto riguarda i colori, ogni playfield ha la sua
palette. I colori 0 e 8 fanno da trasparente, cioe‘ lasciano vedere cosa c’e‘
sotto, allo stesso modo che il trasparente degli sprites. Il colore 0, pero‘,
fa anche da sfondo, nel senso che nelle zone dello schermo in cui entrambi i
playfield sono trasparenti, viene SEMPRE visualizzato il colore 0,
indipendentemente dalla pririta‘ dei due playfield. Per questo motivo, il
colore 0 va comunque settato, mentre e‘ inutile settare il colore 8.
La priorita‘ dei 2 playfield e‘ controllata dal bit 6 del registro BPLCON2
($dff104): se il bit vale 0 il playfield 1 appare sopra al 2, viceversa se il
bit vale 1.
```

23.20 Lezione7v

```
; Lezione7v1.s        PRIORITA‘ SPRITE E PLAYFIELD
;                In questo listato vengono mostrate le priorita‘ tra sprite
;                e playfield. Gli sprites attraversano quattro linee
;                sullo schermo. Ogni volta che attraversano una linea
;                le priorita‘ vengono cambiate mediante la copperlist.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1        ; Nome lib
        jsr         -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC con il solito metodo

        MOVE.L        #PIC,d0
        LEA         BPLPOINTERS,A1
        MOVEQ         #3-1,D1
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #40*256,d0
        addq.w        #8,a1
        dbra         d1,POINTBP

;        Puntiamo gli sprite

        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE1,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE2,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
```

```
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE3,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE4,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE5,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE6,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE7,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!


mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviSprites        ; Muove in basso gli sprites

Aspetta1:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta1


        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse


        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0
```

```
GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; questa routine muove gli 8 sprites in basso:
; gli sprites vengono spostati una volta si ed una no. Per questo
; si usa la variabile flag. Ogni volta che la routine viene eseguita
; la variabile viene cambiata di stato con l'istruzione not:
; se e' a 0, viene posta a $ffff
; se e' a $ffff, viene posta a 0
; se la variabile passa da 0 a $ffff, gli sprite non vengono mossi
; Tutti gli sprites hanno la stessa altezza

Muovisprites:
        not.w       flag
        bne.w       esci

; muove lo sprite 0

        addq.w       #1,altezza
        cmp.w       #300,altezza
        blo.s       no_bordo        ; e' arrivato al bordo inferiore?
        move.w       #$2c,altezza        ; se si lo rimette in alto

no_bordo:
        move.w       altezza(PC),d0

        CLR.B       VHBITS0                ; azzera i bit 8 delle posizioni verticali
        MOVE.b       d0,VSTART0        ; copia i bit da 0 a 7 in VSTART
        BTST.l       #8,D0                ; la posizione e' maggiore di 255 ?
        BEQ.S       NOBIGVSTART        ; se no vai oltre infatti il bit e' stato gia'
                                ; azzerato con la CLR.b VHBITS

        BSET.b       #2,VHBITS0        ; altrimenti metti a 1 il bit 8 della posizione
                                ; verticale di partenza
NOBIGVSTART:
        ADDQ.W       #8,D0                ; Aggiungi la lunghezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b       d0,VSTOP0        ; Muovi i bit da 0 a 7 in VSTOP
        BTST.l       #8,D0                ; la posizione e' maggiore di 255 ?
        BEQ.S       NOBIGVSTOP        ; se no vai oltre infatti il bit e' stato gia'
                                ; azzerato con la CLR.b VHBITS

        BSET.b       #1,VHBITS0        ; altrimenti metti a 1 il bit 8 della posizione
                                ; verticale di fine dello sprite
NOBIGVSTOP:

; copia l'altezza sugli altri sprites

        move.b       vstart0,vstart1        ; copia vstart
        move.w       vstop0,vstop1        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart2        ; copia vstart
        move.w       vstop0,vstop2        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart3        ; copia vstart
        move.w       vstop0,vstop3        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart4        ; copia vstart
```

```
        move.w      vstop0,vstop4       ; copia VSTOP e VHBITS

        move.b      vstart0,vstart5       ; copia vstart
        move.w      vstop0,vstop5       ; copia VSTOP e VHBITS

        move.b      vstart0,vstart6       ; copia vstart
        move.w      vstop0,vstop6       ; copia VSTOP e VHBITS

        move.b      vstart0,vstart7       ; copia vstart
        move.w      vstop0,vstop7       ; copia VSTOP e VHBITS

esci:
        rts

altezza:
        dc.w        $2c
flag:
        dc.w        0


        SECTION     GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81       ; DiwStrt
        dc.w        $90,$2cc1       ; DiwStop
        dc.w        $92,$38             ; DdfStart
        dc.w        $94,$d0             ; DdfStop
        dc.w        $102,0              ; BplCon1
        dc.w        $104,0              ; BplCon2
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000      ; bit 12 acceso!! 3 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo      bitplane
        dc.w $e4,0,$e6,0
        dc.w $e8,0,$ea,0

        dc.w        $180,$000       ; color0        ; sfondo nero
        dc.w        $182,$ff0
        dc.w        $184,$800
        dc.w        $186,$0f0
        dc.w        $188,$ff0
        dc.w        $18a,$f00
        dc.w        $18c,$0f0
        dc.w        $18e,$0f0

        dc.w        $1A2,$F00       ; color17, - COLOR1 degli sprite0/1 -ROSSO
        dc.w        $1A4,$0F0       ; color18, - COLOR2 degli sprite0/1 -VERDE
        dc.w        $1A6,$FF0       ; color19, - COLOR3 degli sprite0/1 -GIALLO

        dc.w        $1AA,$FFF       ; color21, - COLOR1 degli sprite2/3 -BIANCO
        dc.w        $1AC,$0BD       ; color22, - COLOR2 degli sprite2/3 -ACQUA
        dc.w        $1AE,$D50       ; color23, - COLOR3 degli sprite2/3 -ARANCIO
```

```
        dc.w        $1B2,$00F        ; color25, - COLOR1 degli sprite4/5 -BLU
        dc.w        $1B4,$F0F        ; color26, - COLOR2 degli sprite4/5 -VIOLA
        dc.w        $1B6,$BBB        ; color27, - COLOR3 degli sprite4/5 -GRIGIO

        dc.w        $1BA,$8E0        ; color29, - COLOR1 degli sprite6/7 -VERDE CH.
        dc.w        $1BC,$a70        ; color30, - COLOR2 degli sprite6/7 -MARRONE
        dc.w        $1BE,$d00        ; color31, - COLOR3 degli sprite6/7 -ROSSO SC.

; da qui iniziano le istruzioni che cambiano la priorita'
; potete notare che siccome si opera con un playfield normale (non in modo
; dual playfield) il codice di priorita' e' lo stesso per i bit-planes pari
; e per i dispari: per esempio il valore $0009 che e' il primo valore
; che viene scritto in BPLCON2:
;
;        5432109876543210
; $0009=%0000000000001001       potete vedere che:
;
; nei bit da 0 a 2 viene scritto %001
; nei bit da 3 a 5 viene scritto %001, come avevamo detto.
;
; potete verificare che la stessa cosa vale per tutti gli altri valori che
; vengono scritti in BPLCON2


        dc.w        $104,$0000        ; BPLCON2 - all'inizio tutti gli sprite sotto

        dc.w        $7007,$fffe        ; WAIT - attendi la fine della fascia
        dc.w        $104,$0009        ; BPLCON2 - sprites 0,1 sopra e
                                    ; sprites 2,3,4,5,6,7 sotto

        dc.w        $a007,$fffe        ; WAIT - attendi la fine della fascia
        dc.w        $104,$0012        ; BPLCON2 - sprites 0,1,2,3 sopra e
                                    ; sprites 4,5,6,7 sotto

        dc.w        $d007,$fffe        ; WAIT - attendi la fine della fascia
        dc.w        $104,$001b        ; BPLCON2 - sprites 0,1,2,3,4,5 sopra e
                                    ; sprites 6,7 sotto

        dc.w        $ff07,$fffe        ; WAIT - attendi la fine della fascia
        dc.w        $104,$0024        ; BPLCON2 - tutti gli sprites sopra

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;               543210
; NOTA:        $0  = %000000 - tutti gli sprites sotto
;        $9  = %001001 - sprites 0,1 sopra,         2,3,4,5,6,7 sotto
;        $12 = %010010 - sprites 0,1,2,3 sopra,             4,5,6,7 sotto
;        $1b = %011011 - sprites 0,1,2,3,4,5 sopra,         6,7 sotto
;        $24 = %100100 - tutti gli sprites sopra

; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! ************

 ; tabella di riferimento per definire i colori:


 ; per gli sprite 0 ed 1
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (ROSSO)
 ;BINARIO 01=COLORE 2 (VERDE)
 ;BINARIO 11=COLORE 3 (GIALLO)
```

```
MIOSPRITE0:                ; lunghezza 13 linee
VSTART0:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART0:
        dc.b $60        ; Pos. orizzontale (da $40 a $d8)
VSTOP0:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
VHBITS0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite


MIOSPRITE1:                ; lunghezza 13 linee
VSTART1:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART1:
        dc.b $60+14        ; Pos. orizzontale (da $40 a $d8)
VSTOP1:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000010001111
 dc.w        %0011111111111100,%1100000110000011
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0011111111111100,%1100000010000011
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

 ; per gli sprite 2 e 3
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BIANCO)
 ;BINARIO 01=COLORE 2 (ACQUA)
 ;BINARIO 11=COLORE 3 (ARANCIO)

MIOSPRITE2:                ; lunghezza 13 linee
VSTART2:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART2:
        dc.b $60+(14*2)        ; Pos. orizzontale (da $40 a $d8)
VSTOP2:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0011111111111100,%1100001000100011
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0111111111111110,%1000000111000001
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite
```

```
MIOSPRITE3:                ; lunghezza 13 linee
VSTART3:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART3:
        dc.b $60+(14*3)        ; Pos. orizzontale (da $40 a $d8)
VSTOP3:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0111111111111110,%1000000111100001
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

 ; per gli sprite 4 e 5
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BLU)
 ;BINARIO 01=COLORE 2 (VIOLA)
 ;BINARIO 11=COLORE 3 (GRIGIO)

MIOSPRITE4:                ; lunghezza 13 linee
VSTART4:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART4:
        dc.b $60+(14*4)        ; Pos. orizzontale (da $40 a $d8)
VSTOP4:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001001001111
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

MIOSPRITE5:                ; lunghezza 13 linee
VSTART5:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART5:
        dc.b $60+(14*5)        ; Pos. orizzontale (da $40 a $d8)
VSTOP5:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0        ; fine sprite

 ; per gli sprite 6 e 7
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
```

```
 ;BINARIO 10=COLORE 1 (VERDE CHIARO)
 ;BINARIO 01=COLORE 2 (MARRONE)
 ;BINARIO 11=COLORE 3 (ROSSO SCURO)

MIOSPRITE6:                ; lunghezza 13 linee
VSTART6:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART6:
        dc.b $60+(14*6)         ; Pos. orizzontale (da $40 a $d8)
VSTOP6:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w       %0000001111000000,%0111110000111110
 dc.w       %0000111111110000,%1111001111001111
 dc.w       %0011111111111100,%1100001000000011
 dc.w       %0111111111111110,%1000001111000001
 dc.w       %0111111111111110,%1000001001000001
 dc.w       %0011111111111100,%1100001001000011
 dc.w       %0000111111110000,%1111001111001111
 dc.w       %0000001111000000,%0111110000111110
 dc.w       0,0         ; fine sprite

MIOSPRITE7:                ; lunghezza 13 linee
VSTART7:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART7:
        dc.b $60+(14*7)         ; Pos. orizzontale (da $40 a $d8)
VSTOP7:
        dc.b $68        ; $60+13=$6d        ; fine verticale.
        dc.b $00
 dc.w       %0000001111000000,%0111110000111110
 dc.w       %0000111111110000,%1111001111001111
 dc.w       %0011111111111100,%1100000001000011
 dc.w       %0111111111111110,%1000000001000001
 dc.w       %0111111111111110,%1000000001000001
 dc.w       %0011111111111100,%1100000001000011
 dc.w       %0000111111110000,%1111000001001111
 dc.w       %0000001111000000,%0111110000111110
 dc.w       0,0         ; fine sprite

        SECTION         PLANEVUOTO,BSS_C        ; Il bitplane che usiamo

PIC:
        incbin         "priorita.raw"        ; il disegno

        end
```

In questo listato mostriamo come cambiare le priorita‘ degli sprites rispetto ai playfield. Innanzitutto notiamo che gli sprite appaiono sempre al di sopra del colore zero. Per gli altri colori la priorita e‘ controllata dal registro BPLCON2. E‘ possibile controllare la priorita‘ individualmente per i piani pari e per quelli dispari. Questo fatto e‘ molto importante quando si usa il modo dual playfield. Quando, come in questo esempio si usa invece il modo normale si mettono gli stessi livelli di priorita‘ sia per i planes pari che per quelli dispari. Per vedere quali sono i livelli di priorita‘ consultate la lezione.

Per cabiare il livello di priorita‘ piu‘ volte nella stessa schermata, usiamo il copper che ci permette di cambiare priorita‘ quando gli sprites si trovano tra una fascia e l’altra. Ecco i valori per il $dff104 (BPLCON2):

```
              543210
         $0  = %000000 - tutti gli sprites sotto
```

```
$9  = %001001 - sprites 0,1 sopra,         2,3,4,5,6,7 sotto
$12 = %010010 - sprites 0,1,2,3 sopra,             4,5,6,7 sotto
$1b = %011011 - sprites 0,1,2,3,4,5 sopra,         6,7 sotto
$24 = %100100 - tutti gli sprites sopra
```

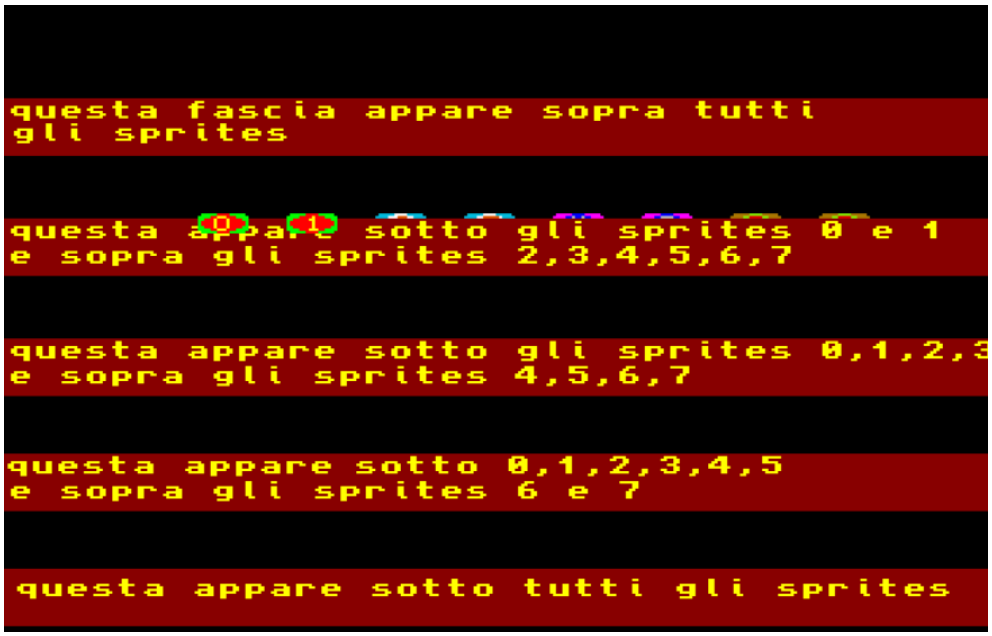


Figura 23.8: Lezione 7v1

Lezione7v2

```
; lezione7v2.s  -  Sprite nel Dual Playfield mode
; In questo esempio mostriamo i vari livelli di priorita‘ per gli sprite
; rispetto ai 2 playfield. Gli sprite si muovono dall’alto in basso.
; Ogni volta che raggiungono il bordo inferiore ripartono dall’alto con un
; diverso livello di priorita‘. Attendere la fine del programma.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)        ; Disable
        lea         GfxName(PC),a1        ; Nome lib
        jsr         -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo le PIC

        MOVE.L        #PIC1,d0        ; puntiamo il playfield 1
        LEA         BPLPOINTERS1,A1
        MOVEQ         #3-1,D1
```

```
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #40*256,d0
        addq.w        #8,a1
        dbra         d1,POINTBP

        MOVE.L        #PIC2,d0        ; puntiamo il playfield 2
        LEA         BPLPOINTERS2,A1
        MOVEQ         #3-1,D1
POINTBP2:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #40*256,d0
        addq.w        #8,a1
        dbra         d1,POINTBP2

;        Puntiamo gli sprite

        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE1,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE2,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE3,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE4,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE5,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE6,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        MOVE.L        #MIOSPRITE7,d0                ; indirizzo dello sprite in d0
        addq.w        #8,a1                        ; prossimi SPRITEPOINTERS
        move.w        d0,6(a1)
        swap         d0
```

```
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

        lea        PriList(PC),a0        ; a0 punta alla lista dei valori di priorita'

        move.w        #$0000,$dff104        ; BPLCON2
                                ; con questo valore gli sprite sono tutti
                                ; sotto entrambi i playfield

aspetta1:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        aspetta1

        bsr.s        MuoviSprites        ; Muove in basso gli sprites

aspetta2:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        aspetta2

        cmp.w        #250,altezza        ; gli sprite hanno raggiunto il bordo basso?
        blo.s        aspetta1        ; no, continua a muoverli

        move.w        #$2c,altezza        ; si. Rimetti gli sprite in alto
        cmp.l        #EndPriList,a0        ; abbiamo terminato i valori di priorita'?
        beq.s        esci                ; se si esci.
        move.w        (a0)+,$dff104        ; se no, metti il prossimo valore in BPLCON2
        bra.s        aspetta1


esci        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine muove gli 8 sprites in basso di un pixel alla volta
; Tutti gli sprites hanno la stessa altezza

Muovisprites:

; muove lo sprite 0

        addq.w        #1,altezza
        move.w        altezza(PC),d0
```

```
        CLR.B       VHBITS0               ; azzera i bit 8 delle posizioni verticali
        MOVE.b       d0,VSTART0        ; copia i bit da 0 a 7 in VSTART
        BTST.l       #8,D0                ; la posizione e‘ maggiore di 255 ?
        BEQ.S       NOBIGVSTART        ; se no vai oltre infatti il bit e‘ stato gia‘
                                ; azzerato con la CLR.b VHBITS

        BSET.b       #2,VHBITS0        ; altrimenti metti a 1 il bit 8 della posizione
                                ; verticale di partenza

NOBIGVSTART:
        ADDQ.W       #8,D0                ; Aggiungi la lunghezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b       d0,VSTOP0        ; Muovi i bit da 0 a 7 in VSTOP
        BTST.l       #8,D0                ; la posizione e‘ maggiore di 255 ?
        BEQ.S       NOBIGVSTOP        ; se no vai oltre infatti il bit e‘ stato gia‘
                                ; azzerato con la CLR.b VHBITS

        BSET.b       #1,VHBITS0        ; altrimenti metti a 1 il bit 8 della posizione
                                ; verticale di fine dello sprite
NOBIGVSTOP:

; copia l’altezza sugli altri sprites

        move.b       vstart0,vstart1        ; copia vstart
        move.w       vstop0,vstop1        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart2        ; copia vstart
        move.w       vstop0,vstop2        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart3        ; copia vstart
        move.w       vstop0,vstop3        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart4        ; copia vstart
        move.w       vstop0,vstop4        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart5        ; copia vstart
        move.w       vstop0,vstop5        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart6        ; copia vstart
        move.w       vstop0,vstop6        ; copia VSTOP e VHBITS

        move.b       vstart0,vstart7        ; copia vstart
        move.w       vstop0,vstop7        ; copia VSTOP e VHBITS

FineMuovisprites:
        rts

altezza:
        dc.w        $2c

; Questa e‘ la lista dei valori di priorita‘. Potete variarla come volete.
; Dopo l’ultimo valore deve esserci pero‘ la label EndPriList
; Questi valori verranno scritti in BPLCON2. Notate che a differenza
; dell’esempio lezione7w1.s qui usiamo uno schermo dual playfield, e
; pertanto possiamo usare per ciascun playfield un diverso livello di
; priorita‘,
; Vi ricordo che tra gli sprite le priorita‘ sono fisse e sono in ordine
; decrescente: lo sprite 0 ha la priorita‘ maggiore, il 7 la minore.

PriList:
        dc.w        $0008        ; %001000 - con questo valore le priorita‘ sono:
```

```
                        ; playfield 1 (sopra tutto)
                        ; sprite 0 e 1
                        ; playfield 2
                        ; sprite 2,3,4,5,6,7 (sotto tutto)

        dc.w        $0010       ; %010000 - con questo valore le priorita' sono:
                        ; playfield 1 (sopra tutto)
                        ; sprite 0,1,2,3
                        ; playfield 2
                        ; sprite 4,5,6,7 (sotto tutto)

        dc.w        $0018       ; %011000 - con questo valore le priorita' sono:
                        ; playfield 1 (sopra tutto)
                        ; sprite 0,1,2,3,4,5
                        ; playfield 2
                        ; sprite 6,7 (sotto tutto)

        dc.w        $0020       ; %100000 - con questo valore le priorita' sono:
                        ; playfield 1 (sopra tutto)
                        ; sprite 0,1,2,3,4,5,6,7
                        ; playfield 2

        dc.w        $0021       ; %100001 - con questo valore le priorita' sono:
                        ; sprite 0 e 1 (sopra tutto)
                        ; playfield 1
                        ; sprite 2,3,4,5,6,7
                        ; playfield 2 (sotto tutto)

        dc.w        $0022       ; %100010 - con questo valore le priorita' sono:
                        ; sprite 0,1,2,3 (sopra tutto)
                        ; playfield 1
                        ; sprite 4,5,6,7
                        ; playfield 2 (sotto tutto)

        dc.w        $0023       ; %100011 - con questo valore le priorita' sono:
                        ; sprite 0,1,2,3,4,5 (sopra tutto)
                        ; playfield 1
                        ; sprite 6,7
                        ; playfield 2 (sotto tutto)

        dc.w        $0024       ; %100100 - con questo valore le priorita' sono:
                        ; sprite 0,1,2,3,4,5,6,7 (sopra tutto)
                        ; playfield 1
                        ; playfield 2 (sotto tutto)
EndPriList:



        SECTION       GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0


        dc.w        $8E,$2c81       ; DiwStrt
        dc.w        $90,$2cc1       ; DiwStop
        dc.w        $92,$38             ; DdfStart
        dc.w        $94,$d0             ; DdfStop
```

```
;        abbiamo tolto il BPLCON2 dalla Copperlist, dato che lo variamo
;        col processore "manualmente".

        dc.w        $102,0                ; BplCon1
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0110011000000000        ; bit 10 acceso = dual playfield
                                        ; uso 6 planes = 8 colori per playfield

BPLPOINTERS1:
        dc.w $e0,0,$e2,0        ;primo bitplane playfield 1 (BPLPT1)
        dc.w $e8,0,$ea,0        ;secondo bitplane playfield 1 (BPLPT3)
        dc.w $f0,0,$f2,0        ;terzo bitplane playfield 1 (BPLPT5)


BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ;primo bitplane playfield 2 (BPLPT2)
        dc.w $ec,0,$ee,0        ;secondo bitplane playfield 2 (BPLPT4)
        dc.w $f4,0,$f6,0        ;terzo bitplane playfield 2 (BPLPT6)

        dc.w        $180,$110        ; palette playfield 1
        dc.w        $182,$005        ; colori da 0 a 7
        dc.w        $184,$a40
        dc.w        $186,$f80
        dc.w        $188,$f00
        dc.w        $18a,$0f0
        dc.w        $18c,$00f
        dc.w        $18e,$080


                                ; palette playfield 2
        dc.w        $192,$367        ; colori da 9 a 15
        dc.w        $194,$0cc         ; il colore 8 e‘ trasparente, non va settato
        dc.w        $196,$a0a
        dc.w        $198,$242
        dc.w        $19a,$282
        dc.w        $19c,$861
        dc.w        $19e,$ff0


        dc.w        $1A2,$F00        ; palette degli sprites
        dc.w        $1A4,$0F0
        dc.w        $1A6,$FF0

        dc.w        $1AA,$FFF
        dc.w        $1AC,$0BD
        dc.w        $1AE,$D50

        dc.w        $1B2,$00F
        dc.w        $1B4,$F0F
        dc.w        $1B6,$BBB

        dc.w        $1BA,$8E0
        dc.w        $1BC,$a70
        dc.w        $1BE,$d00

        dc.w        $FFFF,$FFFE        ; Fine della copperlist
```

```
;        I due playfields


PIC1:        incbin        "dual1.raw"
PIC2:        incbin        "dual2.raw"

; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! ************
MIOSPRITE0:
VSTART0:
        dc.b $60
HSTART0:
        dc.b $60
VSTOP0:
        dc.b $68
VHBITS0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0


MIOSPRITE1:
VSTART1:
        dc.b $60
HSTART1:
        dc.b $60+14
VSTOP1:
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000010001111
 dc.w        %0011111111111100,%1100000110000011
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0111111111111110,%1000000010000001
 dc.w        %0011111111111100,%1100000010000011
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0

MIOSPRITE2:
VSTART2:
        dc.b $60
HSTART2:
        dc.b $60+(14*2)
VSTOP2:
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111000111001111
 dc.w        %0011111111111100,%1100001000100011
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0111111111111110,%1000000111000001
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0
```

```
MIOSPRITE3:
VSTART3:
        dc.b $60
HSTART3:
        dc.b $60+(14*3)
VSTOP3:
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0111111111111110,%1000000111100001
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0011111111111100,%1100000000100011
 dc.w        %0000111111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0


MIOSPRITE4:
VSTART4:
        dc.b $60
HSTART4:
        dc.b $60+(14*4)
VSTOP4:
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001001001111
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0

MIOSPRITE5:
VSTART5:
        dc.b $60
HSTART5:
        dc.b $60+(14*5)
VSTOP5:
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0

MIOSPRITE6:
VSTART6:
        dc.b $60
HSTART6:
        dc.b $60+(14*6)
VSTOP6:
```

```
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0111111111111110,%1000001111000001
 dc.w        %0111111111111110,%1000001001000001
 dc.w        %0011111111111100,%1100001001000011
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0

MIOSPRITE7:
VSTART7:
        dc.b $60
HSTART7:
        dc.b $60+(14*7)
VSTOP7:
        dc.b $68
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001111001111
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0111111111111110,%1000000001000001
 dc.w        %0011111111111100,%1100000001000011
 dc.w        %0000111111110000,%1111000001001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0

        end
```

Questo esempio mostra come funzionano le priorita‘ degli sprite rispetto
ad uno schermo dualplayfield. Per ogni playfield si puo‘ settare un livello
di priorita‘ diverso.
Per questo esempio abbiamo usato una lista di valori di priorita‘.
Una lista in pratica e‘ una serie di valori, come una TABELLA.
Con un registro indirizzi (in questo caso a0) puntiamo al primo valore con
l’istruzione:

```
        lea PriList(PC),a0
```

Ogni volta che viene letto un valore, il registro a0 viene spostato a puntare
il valore successivo, mediante l’indirizzamento indiretto con postincremento,
ovvero:

```
        move.w        (a0)+,$dff104        ; Mettiamo il valore in BPLCON2
```

Quando raggiungiamo l’ultimo valore, a0 viene fatto puntare all’indirizzo
di memoria che segue l’ultimo valore. Questo indirizzo e‘ il valore della
label EndPriList. Quando a0 diventa uguale a EndPriList allora abbiamo
raggiunto la fine della lista, e quindi usciamo dal programma.

Potete cambiare i valori nella lista, sperimentando vari livelli di priorita‘.
Per es. se provate con $0011 vedrete gli sprite 0 e 1 sopra tutti e 2 i
playfield, gli sprite 2 e 3 sopra il playfield 2 e sotto l’uno, mentre gli
altri sprite sotto entrambi i playfield.

NOTA: In questo esempio cambiamo la priorita’ scrivendo direttamente nel
      registro $dff104 (BPLCON2). Questo e’ stato possibile togliendo la
      definizione di tale registro dalla copperlist, ossia la linea:

```
        dc.w        $104,0        ; BPLCON2
```

```
    Se provate a rimettere al suo posto questa istruzione copper, sara'
    annullato l'effetto, proprio perche' ogni fotogramma la copperlist
    viene eseguita, e con essa il BPLCON2 viene azzerato.
    Si puo' decidere dunque di modificare certi registri con la copperlist
    e certi altri direttamente col processore, ma vi consiglio di
    modificarli col copper quando e' possibile, dato che potete sincronizzare
    meglio il momento e la linea video adatti per l'accesso al registro.
```

23.21 Lezione7w

```
; Lezione7w1.s        COLLISIONE TRA SPRITE


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)         ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC con il solito metodo

        MOVE.L        #PIC,d0
        LEA         BPLPOINTERS,A1
        MOVEQ         #2-1,D1
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #40*256,d0
        addq.w        #8,a1
        dbra         d1,POINTBP

;        Puntiamo lo sprite

        LEA         SpritePointers,a1        ; Puntatori in copperlist
        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        add.l         #16,a1

        MOVE.L        #MIOSPRITE2,d0                ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080         ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
```

```
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s         mouse

        bsr.s         MuoviSprite0        ; Muovi l'aereo
        bsr.s         MuoviSprite1        ; Muovi il missile contro l'aereo
        bsr.s         CheckColl         ; Controlla collisione e provvede

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001         ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)         ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)         ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0

OldCop:
        dc.l         0

; Questa routine muove lo sprite dell'aereo in linea retta a sinistra di
; 2 pixel per volta, agendo sul suo HSTART

MuoviSprite0:
        subq.b        #1,HSTART0
        rts

;        -               -               -               -

; Questa routine muove il missile. Lo fa solo se l'aereo e' abbastanza vicino
; da colpirlo. L'aereo e' a tiro quando il suo HSTART e' a $b0.
; Se volete salvare l'aereo provate a far partire il missile alla posizione
; $AA, cioe' troppo presto, oppure alla posizione $c1, ossia troppo tardi.

MuoviSprite1:
        cmp.b         #$b0,HSTART0        ; L'aereo e' a tiro?
        bhi.s         non_a_tiro        ; non partire se l'aereo e' troppo a destra
        subq.b        #1,VSTART2        ; fai salire il missile agendo sia sul
        subq.b        #1,VSTOP2         ; VSTART che sul VSTOP
non_a_tiro:
        rts

;        -               -               -               -

; Questa routine controlla se c'e' collisione. In caso affermativo, cancella
; i due sprites, azzerando i relativi puntatori nella copperlist

CheckColl:
```

```
        move.w        $dff00e,d0        ; legge CLXDAT ($dff00e)
                                ; una lettura di questo registro ne provoca
                                ; anche la cancellazione, per cui conviene
                                ; copiarselo in d0 e fare i test su d0
        btst.l        #9,d0
        beq.s        no_coll                ; se non c'e' collisione salta

        MOVEQ        #0,d0                  ; altrimenti cancella gli sprites
        LEA        SpritePointers,a1 ; puntatore sprite 0
        move.w        d0,6(a1)          ; azzera puntatore sprite 0 in copperlist
        move.w        d0,2(a1)
        add.w        #16,a1                ; puntatore sprite 2
        move.w        d0,6(a1)        ; azzera puntatore sprite 2 in copperlist
        move.w        d0,2(a1)
no_coll:
        rts


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$0024        ; BplCon2 - Sprites davanti i bitplanes
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0010001000000000        ; bit 13 acceso!! 2 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane
        dc.w $e4,0,$e6,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$005        ; color1        ; colore 1 del bitplane
        dc.w        $184,$a40        ; color1        ; colore 2 del bitplane
        dc.w        $186,$f80        ; color1        ; colore 3 del bitplane

        dc.w        $1A2,$06f        ; color17, ossia COLOR1 dello sprite0
        dc.w        $1A4,$0c0        ; color18, ossia COLOR2 dello sprite0
        dc.w        $1A6,$0c0        ; color19, ossia COLOR3 dello sprite0

        dc.w        $1AA,$444        ; color21, ossia COLOR1 dello sprite2
        dc.w        $1AC,$888        ; color22, ossia COLOR2 dello sprite2
        dc.w        $1AE,$0c0        ; color23, ossia COLOR3 dello sprite2

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE0:                ; lunghezza 6 linee
```

```
VSTART0:
        dc.b 180        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART0:
        dc.b $d8        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP0:
        dc.b 186        ; 180+6=186
VHBITS:
        dc.b $00
        dc.w        $0008,$0000
        dc.w        $1818,$0000
        dc.w        $2C28,$1010
        dc.w        $7FF8,$0000
        dc.w        $3FC0,$0000
        dc.w        $01F0,$0000
        dc.w        $0000,$0000

MIOSPRITE2:                ; lunghezza 16 linee
VSTART2:
        dc.b 224        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART2:
        dc.b $86        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP2:
        dc.b 240
        dc.b 0
        dc.w        $0200,$0000
        dc.w        $0200,$0000
        dc.w        $0200,$0000
        dc.w        $0000,$0200
        dc.w        $0000,$0700
        dc.w        $0000,$0700
        dc.w        $0500,$0200
        dc.w        $0200,$0500
        dc.w        $0500,$0200
        dc.w        $0200,$0500
        dc.w        $1540,$0200
        dc.w        $0200,$1DC0
        dc.w        $0000,$1FC0
        dc.w        $0000,$1740
        dc.w        $0500,$0200
        dc.w        $0000,$0000
        dc.w        $0000,$0000

;        Figura della rampa di lancio

PIC:
        incbin        "paesaggio.raw"

        end
```

In questo esempio vediamo come rilevare la collisione tra 2 sprite.
Abbiamo 2 oggetti che si scontrano. Notate che per i 2 oggetti abbiamo usato
gli sprite 0 e 2, cioe‘ 2 sprite che appartengono a 2 gruppi diversi.
Questo fatto innanzitutto ci permette, come abbiamo gia‘ visto, di usare
2 palette diverse per i 2 oggetti, ma e‘ anche necessario per poter
sfruttare la collisione tra sprite. Infatti e‘ possibile rilevare solo
le collisioni tra sprite appartenenti a coppie diverse, e non fra sprite
appartenenti alla stessa coppia. Per rilevare una collisione basta controllare
lo stato di un bit nel registro CLXDAT, come abbiamo gia‘ visto nella lezione.

Quando il bit relativo alla collisione vale 1, allora la collisione si e‘
effettivamente verificata. Il nostro esempio si limita a cancellare i 2
sprite, azzerando i relativi puntatori nella copperlist. Potete migliorare

l'esempio aggiungendo una bella esplosione. E' molto semplice. Innanzitutto disegnatevi uno sprite che rappresenta un'esplosione e aggiungetelo al sorgente (mi raccomando nella SECTION che va nella memoria CHIP).
Poi modificate la routine ChekColl: quando viene rilevata la collisione sostituite alle istruzioni

```
        MOVEQ       #0,d0               ; altrimenti cancella gli sprites
        LEA        SpritePointers,a1        ; puntatore sprite 0
        move.w        d0,6(a1)        ; azzera puntatore sprite 0 in copperlist
        move.w        d0,2(a1)
```

le istruzioni

```
        MOVE.L        #SPRITE_ESPLOSIONE,d0        ; indirizzo sprite esplosione
        LEA        SpritePointers,a1        ; puntatore sprite 0
        move.w        d0,6(a1)        ; modifica puntatore sprite 0 in copperlist
        swap        d0
        move.w        d0,2(a1)
```

in questo modo invece di cancellare lo sprite, sostituirete il disegno dell'aereo con quello dell'esplosione. Dovrete inoltre avere l'accortezza di copiare i byte che controllano la posizione dell'aereo (VSTART0,HSTART0) nei corrispondenti byte di controllo dello sprite dell'esplosione. Questo dovreste saperlo fare da soli, ormai. Dovete fare solo un po' di attenzione a VSTOP: se il disegno dell'esplosione ha un'altezza diversa da quello dell'aereo, non potete semplicemente copiare il VSTOP, ma dovrete aggiustarlo. Nulla di difficile, comunque.

In questo esempio abbiamo volutamente fatto percorrere delle traettorie molto semplici (delle rette) ai 2 sprite, per mostrare meglio il meccanismo della collisione. Potete provare a sostituire alle 2 routine che muovono gli sprite una delle routine che usano tabelle che abbiamo usato negli altri esempi.

Lezione7w2

```
; Lezione7w2.s        COLLISIONE TRA SPRITE DISPARI
; In questo esempio vediamo come rilevare le collisioni degli sprite dispari.
; Questa volta ci sono 2 missili a caccia dell'aereo, e uno dei 2 e' uno
; sprite dispari.
; Se lanciate il programma vedrete pero' che il missile piu' a destra non
; funziona.
; Volete ripararlo ? Leggete il commento finale!


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC con il solito metodo

        MOVE.L        #PIC,d0
        LEA        BPLPOINTERS,A1
        MOVEQ        #2-1,D1
```

```
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #40*256,d0
        addq.w        #8,a1
        dbra         d1,POINTBP

;        Puntiamo lo sprite

        LEA         SpritePointers,a1        ; Puntatori in copperlist
        MOVE.L        #MIOSPRITE0,d0               ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        add.l         #16,a1                      ; puntatore sprite 2
        MOVE.L        #MIOSPRITE2,d0               ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        add.l         #8,a1                       ; puntatore sprite 3
        MOVE.L        #MIOSPRITE3,d0               ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s         mouse

        bsr.s         MuoviSprite0        ; Muovi l'aereo
        bsr.s         MuoviSprite2        ; Muovi il missile 1 contro l'aereo
        bsr.s         MuoviSprite3        ; Muovi il missile 2 contro l'aereo
        bsr.w         CheckColl        ; Controlla collisione e provvede

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001        ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
```

```
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine muove lo sprite dell'aereo in linea retta di 2 pixel per volta

MuoviSprite0:
        subq.b       #1,HSTART0
        rts


; Questa routine muove il missile. Lo fa solo se l'aereo e' abbastanza vicino
; da colpirlo.

MuoviSprite2:
        cmp.b       #$b0,HSTART0
        bhi.s       non_a_tiro2        ; non partire se l'aereo e' troppo a destra
        subq.b       #1,VSTART2
        subq.b       #1,VSTOP2
non_a_tiro2:
        rts


; Questa routine muove il missile. Lo fa solo se l'aereo e' abbastanza vicino
; da colpirlo.

MuoviSprite3:
        cmp.b       #$d0,HSTART0
        bhi.s       non_a_tiro3        ; non partire se l'aereo e' troppo a destra
        subq.b       #1,VSTART3
        subq.b       #1,VSTOP3
non_a_tiro3:
        rts

; Questa routine controlla se c'e' collisione. In caso affermativo, cancella
; i due sprites che hanno scontrato, azzerando i relativi puntatori nella
; copperlist. Per distinguere quale dei 2 missili ha colpito l'aereo,
; vengono controllate le posizioni. Infatti in questo esempio i missili
; possono colpire l'aereo solo dal basso; quindi se un missile si trova
; piu' in alto dell'aereo NON PUO' averlo colpito. In questo modo riusciamo
; a capire quale dei 2 missili ha colpito l'aereo.

CheckColl:
        move.w       $dff00e,d0        ; legge CLXDAT ($dff00e)
                                 ; una lettura di questo registro ne provoca
                                 ; anche la cancellazione, per cui conviene
                                 ; copiarselo in d0 e fare i test su d0
        btst.l       #9,d0
        beq.s       no_coll                ; se non c'e' collisione salta

        MOVEQ       #0,d0                  ; altrimenti cancella l'aereo
        LEA        SpritePointers,a1 ; puntatore sprite 0
        move.w       d0,6(a1)          ; azzera puntatore sprite 0 in copperlist
        move.w       d0,2(a1)

; ora dobbiamo capire quale dei 2 missili ha colpito l'aereo.
; controlliamo l'altezza del missile piu' a destra: se si trova piu'
; in alto NON ha colpito l'aereo, altrimenti e' stato lui
```

```
        move.b        VSTART0,d1        ; legge l'altezza dell'aereo
        cmp.b        VSTART3,d1        ; confronta con l'altezza del missile a destra
        bhi.s        spr2_coll        ; se l'aereo e' piu' in basso
                                ; (quindi VSTART0 e' MAGGIORE di VSTART3)
                                ; la collisione e' causata dallo sprite 2

        LEA        SpritePointer3,a1 ; altrimenti cancella sprite 3
        move.w        d0,6(a1)          ; azzera puntatore sprite 3 in copperlist
        move.w        d0,2(a1)
        bra.s        no_coll

spr2_coll:
        LEA        SpritePointer2,a1 ; cancella sprite 2
        move.w        d0,6(a1)          ; azzera puntatore sprite 2 in copperlist
        move.w        d0,2(a1)
no_coll:
        rts



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0
SpritePointer2:
        dc.w        $128,0,$12a,0
SpritePointer3:
        dc.w        $12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

                    ; 5432109876543210
        dc.w        $98,%0000000000000000        ; CLXCON  $dff098

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$0024        ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0010001000000000        ; 2 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane
        dc.w $e4,0,$e6,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$005        ; color1        ; colore 1 del bitplane
        dc.w        $184,$a40        ; color1        ; colore 2 del bitplane
        dc.w        $186,$f80        ; color1        ; colore 3 del bitplane

        dc.w        $1A2,$06f        ; color17, ossia COLOR1 dello sprite0
        dc.w        $1A4,$0c0        ; color18, ossia COLOR2 dello sprite0
        dc.w        $1A6,$0c0        ; color19, ossia COLOR3 dello sprite0

        dc.w        $1AA,$444        ; color21, ossia COLOR1 dello sprite2
```

```
        dc.w        $1AC,$888        ; color22, ossia COLOR2 dello sprite2
        dc.w        $1AE,$0c0        ; color23, ossia COLOR3 dello sprite2

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE0:                ; lunghezza 6 linee
VSTART0:
        dc.b 180        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART0:
        dc.b $d8        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP0:
        dc.b 186        ; 180+6=186
VHBITS:
        dc.b $00
        dc.w        $0008,$0000
        dc.w        $1818,$0000
        dc.w        $2C28,$1010
        dc.w        $7FF8,$0000
        dc.w        $3FC0,$0000
        dc.w        $01F0,$0000
        dc.w        $0000,$0000

MIOSPRITE2:                ; lunghezza 16 linee
VSTART2:
        dc.b 224        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART2:
        dc.b $86        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP2:
        dc.b 240
        dc.b 0
        dc.w        $0200,$0000
        dc.w        $0200,$0000
        dc.w        $0200,$0000
        dc.w        $0000,$0200
        dc.w        $0000,$0700
        dc.w        $0000,$0700
        dc.w        $0500,$0200
        dc.w        $0200,$0500
        dc.w        $0500,$0200
        dc.w        $0200,$0500
        dc.w        $1540,$0200
        dc.w        $0200,$1DC0
        dc.w        $0000,$1FC0
        dc.w        $0000,$1740
        dc.w        $0500,$0200
        dc.w        $0000,$0000
        dc.w        $0000,$0000

MIOSPRITE3:                ; lunghezza 16 linee
VSTART3:
        dc.b 224        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART3:
        dc.b $a6        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP3:
        dc.b 240
        dc.b 0
        dc.w        $0200,$0000
        dc.w        $0200,$0000
        dc.w        $0200,$0000
```

```
        dc.w        $0000,$0200
        dc.w        $0000,$0700
        dc.w        $0000,$0700
        dc.w        $0500,$0200
        dc.w        $0200,$0500
        dc.w        $0500,$0200
        dc.w        $0200,$0500
        dc.w        $1540,$0200
        dc.w        $0200,$1DC0
        dc.w        $0000,$1FC0
        dc.w        $0000,$1740
        dc.w        $0500,$0200
        dc.w        $0000,$0000
        dc.w        $0000,$0000

;        figura della rampa di lancio

PIC:
        incbin        "paesaggio.raw"

        end
```

Come abbiamo visto nella lezione, il registro CLXDAT ci permette di rilevare collisioni tra gruppi di sprite e non tra i singoli sprite. In questo esempio vediamo come risolvere il problema. Vi ricordo che mentre le collisioni degli sprite pari (cioe‘ sprite 0,2,4,6) sono sempre abiitate, quelle degli sprite dispari (1,3,5,7) devono essere abilitate mediante dei bit di controllo del registro CLXCON ($dff098). Ogni sprite dispari ha un proprio bit di controllo e pertanto puo‘ essere abilitato indipendentemente dagli altri. Se provate ad eseguire il nostro esempio, noterete che il missile piu‘ a destra non funziona. Cio‘ avviene proprio perche‘ il missile piu‘ a destra e‘ uno sprite dispari (sprite 3) ed e‘ disabilitato. Infatti nella copperlist potete trovare l’istruzione:

```
                    ; 5432109876543210
        dc.w         $98,%0000000000000000
```

Che disabilita per le collisioni tutti gli sprite dispari (per il significato preciso di tutti i bit guardate nella lezione). Per abilitare lo sprite 3, si deve settare a 1 il bit 13, cioe‘ modificare l’istruzione copper come segue:

```
                    ; 5432109876543210
        dc.w         $98,%0010000000000000
```

Provate ad eseguire l’esempio, e vedrete che ora il missile funziona!

Un altro problema delle collisioni e‘ che il registro CLXDAT rivela collisioni tra gruppi di sprite, e non tra i singoli sprite. Nel nostro esempio ci sono i 2 missili che appartengono allo stesso gruppo. Quindi quando c’e‘ una collisione non possiamo sapere quale dei 2 missili ha colpito l’aereo leggendo il registro CLXDAT. Per farlo il metodo piu‘ usato e‘ quello di controllare le posizioni degli sprite. In questo esempio particolarmente semplice, basta controllare se lo sprite piu‘ a destra sia o meno al di sopra dell’aereo, come abbiamo spiegato piu‘ in dettaglio nel commento alla routine CheckColl. In situazioni piu‘ complesse con gli sprite che si muovono in piu‘ direzioni e‘ necessario fare dei controlli piu‘ accurati, basandosi sia sulle posizioni verticali che su quelle orizzontali. Il principio e‘ comunque sempre lo stesso.

Potete verificare che la nostra routine individua sempre il missile giusto modificando la posizione di partenza dello sprite piu‘ a destra.
Il valore iniziale di HSTART3 e‘ $a6 e garantisce al misssile di colpire l’aereo. Sostituite $a6 con $b6. Se eseguite l’esempio vedrete che il missile si trova troppo a destra e pertanto manchera‘ l’aereo. Ma non temete!

```
Il secondo fara‘ comunque centro!
```

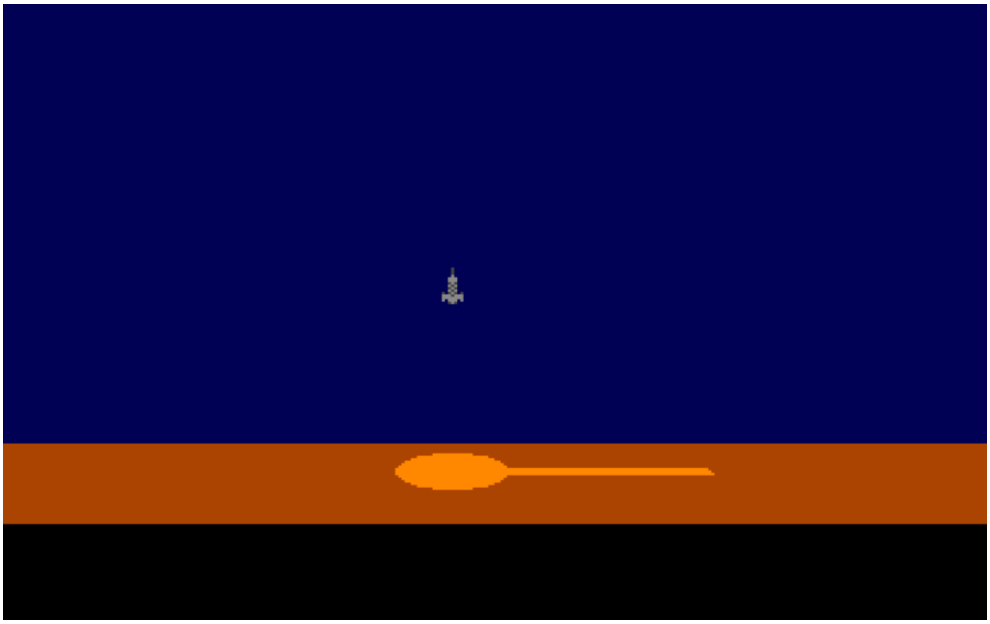


Figura 23.9: Lezione 7w2

23.22 Lezione7x

```
; Lezione7x1.s        COLLISIONE TRA SPRITE E PLAYFIELD
;                In questo esempio c’e‘ uno sprite che attraversa rettangoli
;                di diversi colori. Quando lo sprite tocca un determinato
;                colore, si accende un rivelatore.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr         -$78(a6)         ; Disable
        lea         GfxName(PC),a1         ; Nome lib
        jsr         -$198(a6)         ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC con il solito metodo

        MOVE.L        #PIC,d0
        LEA         BPLPOINTERS,A1
        MOVEQ         #3-1,D1
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
```

```
        swap        d0
        ADD.L        #40*256,d0
        addq.w        #8,a1
        dbra        d1,POINTBP

;        Puntiamo lo sprite

        LEA        SpritePointers,a1        ; Puntatori in copperlist
        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        add.l        #16,a1

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        MuoviSprite0        ; Muovi l'aereo
        bsr.s        CheckColl        ; Controlla collisione e provvede

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine muove lo sprite in linea retta di 2 pixel per volta

MuoviSprite0:
        subq.b        #1,HSTART0
        rts


; Questa routine controlla se c'e' collisione.
; In caso affermativo, "accende" il rivelatore di collisione.
```

```
; Il rivelatore e‘ semplicemente un rettangolo colorato con il COLORE 7.
; Modificando nella copperlist il valore assunto dal registro COLOR07,
; si provoca l’accensione (rosso) o lo spegnimento (grigio) del rivelatore.

CheckColl:
        move.w        $dff00e,d0        ; legge CLXDAT ($dff00e)
                                  ; una lettura di questo registro ne provoca
                                  ; anche la cancellazione, per cui conviene
                                  ; copiarselo in d0 e fare i test su d0
        move.w        d0,d7
        btst.l        #1,d0               ; il bit 1 indica la collisione tra sprite 0
                                  ; e playfield
        beq.s        no_coll               ; se non c’e‘ collisione salta

si_coll:
        move.w        #$f00,rileva_collisione ; "accende" il rivelatore (COLOR07)
                                        ; modificando la copperlist (rosso)
        bra.s        exitColl               ; esci

no_coll:
        move.w        #$555,rileva_collisione ; "spegne" il rivelatore (COLOR07)
                                        ; modificando la copperlist (grigio)
exitColl:
        rts



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0


; Questo e‘ il registro CLXCON (controlla il modo di rilevamento)

; i bit da 0 a 5 sono i valori che devono essere assunti dai plane
; i bit da 6 a 11 indicano quali planes sono abilitati alle collisioni
; i bit da 12 a 15 indicano quali degli sprite dispari sono abilitati
; al rilevamento delle collisioni.
                    ;5432109876543210
        dc.w        $98,%0000000111000011        ; CLXCON

; Questi valori indicano che i planes 1,2 e 3 sono attivi per le collisioni, e
; che viene segnalata collisione quando lo sprite si sovrappone ad un pixel
; che ha:        plane 1 = 1
;                 plane 2 = 1
;                plane 3 = 0


        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1

        dc.w        $104,$0024        ; BplCon2 - mette tutti gli sprite davanti ai
                                  ; playfield
```

```
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; 3 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0
        dc.w $e4,0,$e6,0
        dc.w $e8,0,$ea,0

; colori bitplanes
        dc.w        $180,$000       ; color0        ; sfondo nero
        dc.w        $182,$620
        dc.w        $184,$fff
        dc.w        $186,$e00
        dc.w        $188,$808
        dc.w        $18a,$f4a
        dc.w        $18c,$aaa
        dc.w        $18e        ; color07 - il valore caricato in questo registro viene
                    ; scritto dalla routine ChekColl a seconda del verifi-
                    ; carsi o meno di una collisione.
rileva_collisione:
        dc.w        0        ; IN QUESTO PUNTO la routine CheckColl modifica
                    ; la copper list scrivendo il colore giusto.

; colori sprite
        dc.w        $1A2,$00f       ; color17, ossia COLOR1 dello sprite0
        dc.w        $1A4,$0c0       ; color18, ossia COLOR2 dello sprite0
        dc.w        $1A6,$0c0       ; color19, ossia COLOR3 dello sprite0

        dc.w        $FFFF,$FFFE       ; Fine della copperlist


; ************ Ecco lo sprite: OVVIAMENTE deve essere in CHIP RAM! ************

MIOSPRITE0:                ; lunghezza 6 linee
VSTART0:
        dc.b 200        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART0:
        dc.b $d8        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP0:
        dc.b 206        ; 200+6=206
VHBITS:
        dc.b $00
        dc.w        $0008,$0000
        dc.w        $1818,$0000
        dc.w        $2C28,$1010
        dc.w        $7FF8,$0000
        dc.w        $3FC0,$0000
        dc.w        $01F0,$0000
        dc.w        $0000,$0000

PIC:
        incbin       "collpic.raw"

        end
```

In questo esempio mostriamo come rilevare le collisioni tra sprite e plafield.
Come abbiamo gia‘ visto nella lezione, si usano i due registri CLXDAT e CLXCON.
CLXDAT serve semplicemente per sapere se si e‘ verificata una collisione e si
usa esattamente come nel caso di collisione tra 2 sprite (solo che ovviamente

si usa un bit diverso). L'uso di CLXCON, invece e' piu' complesso. Vediamolo
bene studiando il nostro esempio. Abbiamo scritto nella copper list:

```
                ;5432109876543210
        dc.w    $98,%0000000111000011        ; CLXCON
```

I bit da 6 a 11 indicano quali planes sono abilitati alle collisioni. Nel
nostro esempio sono abilitati i planes 1,2,3 (i planes visualizzati). I bit
da 0 a 5 indicano invece i valori che devono essere assunti dai plane affinche'
si verifichi la collisione. Nel nostro esempio la collisione c'e' se i 3 planes
abilitati assumono i seguenti valori: plane 3 = 0, plane 2 = 1, plane 1 = 1,
ovvero la sequenza %011=3. Quidi viene rilevata la collisione tra lo sprite
e il colore 3. Notate che non ha interesse il valore che devono assumere
i planes 4,5 e 6 in quanto sono disabilitati.

Modificate ora la copper list come segue:

```
                ;5432109876543210
        dc.w    $98,%0000000111000010        ; CLXCON
```

Ora i planes abilitati sono sempre 1,2 e 3 ma il valore che devono assumere
e' pari alla sequenza %010, ovvero il colore 2. Potete verificarlo lanciando
il programma. Funziona allo stesso modo per gli altri colori.

E se volessimo rilevare la collisione con piu' di 2 colori ?
In certi casi e' possibile farlo non abilitando tutti i piani visualizzati.
Modificate la copper list come segue:

```
                ;5432109876543210
        dc.w    $98,%0000000110000010        ; CLXCON
```

A differenza di prima, abbiamo abilitato per il rilevamento collisioni
solo i planes 2 e 3. Questo significa che il valore del plane 1 non ha effetto
sul rilevamento delle collisioni. E' necessario solamente che:
plane 3 = 0 e plane 2 = 1. Poiche' cio' si verifica sia per la sequenza
binaria %010 sia per la sequenza %011, entrambe daranno luogo ad una
collisione. In questo modo la collisione avverra' per il colore 2 (%010)
e per il colore 3 (%011).

Vediamo un'altro esempio. Modificate la copper list come segue:

```
                ;5432109876543210
        dc.w    $98,%0000000001000001        ; CLXCON
```

Ora e' abilitato solo il plane 1, e la collisione si verifica quando il
plane 1 = 1. Questo accade per tutti i colori dispari. Si ha infatti:

```
        %001        colore 1
        %011        colore 3
        %101        colore 5
        %111        colore 7
```

In tutte e quattro queste combinazioni il plane 1 vale 1.

Tutto cio' che abbiamo detto e' valido anche nel caso di un numero di planes
visualizzati diverso da 3.

Lezione7x2

```
; lezione7x2.s         - Collsioni Sprite in Dual Playfield mode
; In questo esempio mostriamo le collisioni tra uno sprite e i due playfield.
; Lo sprite si muove dall'alto in basso. Se viene rilevata una collisione,
; viene cambiato il colore di sfondo (rosso o verde a seconda di cosa e'
```

```
; in collisione).

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

; Usiamo 2 planes per ogni playfield

;        Puntiamo le PIC

        MOVE.L        #PIC1,d0        ; puntiamo il playfield 1
        LEA        BPLPOINTERS1,A1
        MOVEQ        #2-1,D1
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*256,d0
        addq.w        #8,a1
        dbra        d1,POINTBP

        MOVE.L        #PIC2,d0        ; puntiamo il playfield 2
        LEA        BPLPOINTERS2,A1
        MOVEQ        #2-1,D1
POINTBP2:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*256,d0
        addq.w        #8,a1
        dbra        d1,POINTBP2

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE0,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

        move.w        #$0024,$dff104        ; BPLCON2
                        ; con questo valore gli sprite sono tutti
                        ; sopra entrambi i playfield

aspetta1:
        cmp.b        #$ff,$dff006        ; Linea 255?
        bne.s        aspetta1
aspetta11:
```

```
        cmp.b       #$ff,$dff006        ; Ancora Linea 255?
        beq.s       aspetta11

        btst        #6,$bfe001
        beq.s       esci

        bsr.s       MuoviSprite         ; Muove in basso lo sprite
        bsr.w       CheckColl           ; Controlla collisione e provvede

        bra.s       aspetta1

esci        move.l      OldCop(PC),$dff080      ; Puntiamo la cop di sistema
        move.w      d0,$dff088              ; facciamo partire la vecchia cop

        move.l      4.w,a6
        jsr         -$7e(a6)        ; Enable
        move.l      gfxbase(PC),a1
        jsr         -$19e(a6)       ; Closelibrary
        rts

;       Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine muove lo sprite 0 in basso di un pixel ogni 2 fotogrammi
; Viene usato un flag.

Muovisprite:
        not.w       flag
        beq.s       FineMuovisprite

        addq.w      #1,altezza
        cmp.w       #300,altezza        ; e' arrivato al bordo inferiore?
        blo.s       no_bordo
        move.w      #$2c,altezza        ; se si, rimetti lo sprite in alto
no_bordo:
        move.w      altezza(PC),d0
        CLR.B       VHBITS0             ; azzera i bit 8 delle posizioni verticali
        MOVE.b      d0,VSTART0          ; copia i bit da 0 a 7 in VSTART
        BTST.l      #8,D0               ; la posizione e' maggiore di 255 ?
        BEQ.S       NOBIGVSTART         ; se no vai oltre, infatti il bit e' stato gia'
                                ; azzerato con la CLR.b VHBITS

        BSET.b      #2,VHBITS0          ; altrimenti metti a 1 il bit 8 della posizione
                                ; verticale di partenza
NOBIGVSTART:
        ADDQ.w      #8,D0               ; Aggiungi la lunghezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b      d0,VSTOP0           ; Muovi i bit da 0 a 7 in VSTOP
        BTST.l      #8,D0               ; la posizione e' maggiore di 255 ?
        BEQ.S       NOBIGVSTOP          ; se no vai oltre infatti il bit e' stato gia'
                                ; azzerato con la CLR.b VHBITS
        BSET.b      #1,VHBITS0          ; altrimenti metti a 1 il bit 8 della posizione
                                ; verticale di fine dello sprite
NOBIGVSTOP:
```

```
FineMuovisprite:
        rts


; Questa routine controlla se c'e' collisione.
; In caso affermativo, cambia il colore dello sfondo modificando nella copper
; list il valore assunto dal registro COLOR00, rosso o verde.

CheckColl:
        move.w        $dff00e,d0        ; legge CLXDAT ($dff00e)
                                  ; una lettura di questo registro ne provoca
                                  ; anche la cancellazione, per cui conviene
                                  ; copiarselo in d0 e fare i test su d0
        btst.l        #1,d0               ; il bit 1 indica la collisione tra sprite 0
                                  ; e playfield 1
        beq.s         no_coll1            ; se non c'e' collisione salta

        move.w        #$f00,rileva_collisione ; "accende" il rivelatore (color0)
                                      ; modificando la copperlist (rosso)
        bra.s         exitColl            ; esci

no_coll1:
        btst.l        #5,d0               ; il bit 5 indica la collisione tra sprite 0
                                  ; e playfield 2
        beq.s         no_coll2            ; se non c'e' collisione salta
        move.w        #$0f0,rileva_collisione ; "accende" il rivelatore (color0)
                                      ; modificando la copperlist (verde)
        bra.s         exitColl            ; esci

no_coll2:
        move.w        #$000,rileva_collisione ; "spegne" il rivelatore (color0)
                                      ; modificando la copperlist (nero)
exitColl:
        rts

flag:
        dc.w        0
altezza:
        dc.w        $2c


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0              ; BplCon1
        dc.w        $108,0              ; Bpl1Mod
        dc.w        $10a,0              ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0100011000000000        ; bit 10 acceso = dual playfield
                                      ; uso 4 planes = 4 colori per playfield
```

```
BPLPOINTERS1:
        dc.w $e0,0,$e2,0        ;primo bitplane playfield 1 (BPLPT1)
        dc.w $e8,0,$ea,0        ;secondo bitplane playfield 1 (BPLPT3)


BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ;primo bitplane playfield 2 (BPLPT2)
        dc.w $ec,0,$ee,0        ;secondo bitplane playfield 2 (BPLPT4)

; Questo e' il registro CLXCON (controlla il modo di rilevamento)

; i bit da 0 a 5 sono i valori che devono essere assunti dai plane
; i bit da 6 a 11 indicano quali planes sono abilitati alle collisioni
; i bit da 12 a 15 indicano quali degli sprite dispari sono abilitati
; al rilevamento delle collisioni.

                ;5432109876543210
        dc.w    $98,%0000001111001011        ; CLXCON

; Questi valori indicano che i planes 1,2,3,4 sono attivi per le collisioni.
; Viene segnalata collisione con il playfield 1 quando lo sprite si sovrappone
; ad un pixel che ha:        plane 1 = 1 (bit 0)
;                       plane 3 = 0 (bit 2)
; cioe' con il colore 1 del playfield 1

; viene segnalata collisione con il playfield 2 quando lo sprite si sovrappone
; ad un pixel che ha:        plane 2 = 1 (bit 1)
;                       plane 4 = 1 (bit 3)
; cioe' con il colore 3 del playfield 2


        dc.w    $180    ; COLOR00
rileva_collisione:
        dc.w    0        ; IN QUESTO PUNTO la routine CheckColl modifica
                ; la copper list scrivendo il colore giusto.

                        ; palette playfield 1
        dc.w    $182,$005       ; colori da 0 a 7
        dc.w    $184,$a40
        dc.w    $186,$f80
        dc.w    $188,$f00
        dc.w    $18a,$0f0
        dc.w    $18c,$00f
        dc.w    $18e,$080


                        ; palette playfield 2
        dc.w    $192,$367       ; colori da 9 a 15
        dc.w    $194,$0cc        ; il colore 8 e' trasparente, non va settato
        dc.w    $196,$a0a
        dc.w    $198,$242
        dc.w    $19a,$282
        dc.w    $19c,$861
        dc.w    $19e,$ff0


        dc.w    $1A2,$F00       ; palette degli sprites
        dc.w    $1A4,$0F0
        dc.w    $1A6,$FF0

        dc.w    $1AA,$FFF
        dc.w    $1AC,$0BD
```

```
        dc.w        $1AE,$D50

        dc.w        $1B2,$00F
        dc.w        $1B4,$F0F
        dc.w        $1B6,$BBB

        dc.w        $1BA,$8E0
        dc.w        $1BC,$a70
        dc.w        $1BE,$d00

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        I due playfields

PIC1:        incbin        "colldual1.raw"
PIC2:        incbin        "colldual2.raw"

; ************ Ecco lo sprite: OVVIAMENTE in CHIP RAM! ************
MIOSPRITE0:
VSTART0:
        dc.b $2c
HSTART0:
        dc.b $80
VSTOP0:
        dc.b $2c+8
VHBITS0
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0111111111111110,%1000010001000001
 dc.w        %0011111111111100,%1100010001000011
 dc.w        %0000111111110000,%1111001110001111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0


        end
```

Questo esempio mostra come funzionano le collisioni tra sprite e i playfield di uno schermo (dualplayfield). Si controllano indipendentemente le collisioni con i 2 playfield usando 2 diversi bit del registro CLXDAT. Nel nostro esempio (usiamo lo sprite 0), il bit 1 controlla la collisione con il playfield 1 (piani dispari) e il bit 5 la collisione con il playfield 2 (piani pari).
Per quanto il registro CLXCON, fuunziona tutto come per il caso di schermo normale:
i bit da 0 a 5 sono i valori che devono essere assunti dai plane
i bit da 6 a 11 indicano quali planes sono abilitati alle collisioni
i bit da 12 a 15 indicano quali degli sprite dispari sono abilitati al rilevamento delle collisioni.
Rimane sempre possibile non abilitare alcuni planes per rilevare piu‘ colori contemporaneamente, come abbiamo illustrato con l’esempio lezione7w2.
Potete provare modificando nella copperlist il valore assegnato a CLXCON.

Lezione7x3

```
; lezione7x3.s        - Collsioni tra playfield in Dual Playfield mode
```

Figura 23.10: Lezione 7x2

```
; In questo esempio mostriamo le collisioni tra i due playfield.
; Il playfield 1 si muove dall'alto in basso.
; Se il colore 3 del playfield 1 si sovrappone al colore 1 del playfield 2
; viene rilevata una collisione e viene cambiato il colore di sfondo

        SECTION        CiriCop,CODE

Inizio:
        move.l         4.w,a6                ; Execbase
        jsr           -$78(a6)         ; Disable
        lea           GfxName(PC),a1         ; Nome lib
        jsr           -$198(a6)         ; OpenLibrary
        move.l          d0,GfxBase
        move.l          d0,a6
        move.l          $26(a6),OldCop        ; salviamo la vecchia COP

; Usiamo 2 planes per ogni playfield

;        Puntiamo le PIC

        MOVE.L          #PIC1,d0        ; puntiamo il playfield 1
        LEA           BPLPOINTERS1,A1
        MOVEQ           #2-1,D1
POINTBP:
        move.w          d0,6(a1)
        swap           d0
        move.w          d0,2(a1)
        swap           d0
        ADD.L           #40*256,d0
        addq.w          #8,a1
        dbra           d1,POINTBP
```

```
        MOVE.L        #PIC2,d0        ; puntiamo il playfield 2
        LEA        BPLPOINTERS2,A1
        MOVEQ        #2-1,D1
POINTBP2:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*256,d0
        addq.w        #8,a1
        dbra        d1,POINTBP2

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

        move.w        #$0024,$dff104        ; BPLCON2
                          ; con questo valore gli sprite sono tutti
                          ; sopra entrambi i playfield

aspetta1:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        aspetta1
aspetta11:
        cmpi.b        #$ff,$dff006        ; Ancora Linea 255?
        beq.s        aspetta11

        btst        #6,$bfe001
        beq.s        esci

        bsr.s        MuoviCopper        ; Muove il playfield 1
        bsr.w        CheckColl        ; Controlla collisione e provvede

        bra.s        aspetta1

esci        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


; Questa routine muove un playfield in basso. E‘ la stessa della lezione 5
; solo che spostiamo solo il playfield 1, cioe‘ solo i bitplanes dispari.

MuoviCopper:
        LEA        BPLPOINTERS1,A1        ; Con queste 4 istruzioni preleviamo dalla
```

```
        move.w        2(a1),d0        ; copperlist l'indirizzo dove sta puntando
        swap        d0                ; attualmente il $dff0e0 e lo poniamo
        move.w        6(a1),d0        ; in d0 - il contrario della routine che
                                ; punta i bitplanes! Qua invece di mettere
                                ; l'indirizzo lo prendiamo!!!

        TST.B        SuGiu                ; Dobbiamo salire o scendere? se SuGiu e'
                                ; azzerata, (cioe' il TST verifica il BEQ)
                                ; allora saltiamo a VAIGIU, se invece e' a $FF
                                ; (se cioe' questo TST non e' verificato)
                                ; continuiamo salendo (facendo dei sub)
        beq.w        VAIGIU
        cmp.l        #PIC1-(40*90),d0        ; siamo arrivati abbastanza in ALTO?
        beq.s        MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        sub.l        #40,d0                ; sottraiamo 40, ossia 1 linea, facendo
                                ; scorrere in BASSO la figura
        bra.s        Finito

MettiGiu:
        clr.b        SuGiu                ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        bra.s        Finito                ; fara' saltare alla routine VAIGIU

VAIGIU:
        cmpi.l        #PIC1+(40*30),d0        ; siamo arrivati abbastanza in BASSO?
        beq.s        MettiSu                ; se si, siamo in fondo e dobbiamo risalire
        add.l        #40,d0                ; Aggiungiamo 40, ossia 1 linea, facendo
                                ; scorrere in ALTO la figura
        bra.s        finito

MettiSu:
        move.b        #$ff,SuGiu        ; Quando la label SuGiu non e' a zero,
        rts                        ; significa che dobbiamo risalire.

Finito:                                ; PUNTIAMO I PUNTATORI BITPLANES
        LEA        BPLPOINTERS1,A1        ; puntatori nella COPPERLIST
        MOVEQ        #1,D1                ; numero di bitplanes -1 (qua sono 2)
POINTBP3:
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 3412 > 1234)
        ADD.L        #40*256,d0        ; + lunghezza bitplane -> prossimo bitplane
        addq.w        #8,a1                ; andiamo ai prossimi bplpointers nella COP
        dbra        d1,POINTBP3        ; Rifai D1 volte POINTBP (D1=num of bitplanes)
        rts


;        Questo byte, indicato dalla label SuGiu, e' un FLAG.

SuGiu:
        dc.b        0,0


; Questa routine controlla se c'e' collisione.
; In caso affermativo, cambia il colore dello sfondo
; modificando nella copper list il valore assunto dal registro COLOR00.

CheckColl:
        move.w        $dff00e,d0        ; legge CLXDAT ($dff00e)
                                ; una lettura di questo registro ne provoca
                                ; anche la cancellazione, per cui conviene
                                ; copiarselo in d0 e fare i test su d0
```

```
        btst.l        #0,d0               ; il bit 0 indica la collisione tra playfield
        beq.s        no_coll               ; se non c'e' collisione salta

        move.w        #$f00,rileva_collisione ; "accende" il rivelatore (color0)
                                    ; modificando la copperlist (rosso)
        bra.s        ExitColl

no_coll:
        move.w        #$000,rileva_collisione ; "spegne" il rivelatore (color0)
                                    ; modificando la copperlist (nero)
ExitColl:
        rts

flag:
        dc.w        0
altezza:
        dc.w        $2c



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0100011000000000        ; bit 10 acceso = dual playfield
                                    ; uso 4 planes = 4 colori per playfield

BPLPOINTERS1:
        dc.w $e0,0,$e2,0        ;primo bitplane playfield 1 (BPLPT1)
        dc.w $e8,0,$ea,0        ;secondo bitplane playfield 1 (BPLPT3)


BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ;primo bitplane playfield 2 (BPLPT2)
        dc.w $ec,0,$ee,0        ;secondo bitplane playfield 2 (BPLPT4)

; Questo e' il registro CLXCON (controlla il modo di rilevamento)

; i bit da 0 a 5 sono i valori che devono essere assunti dai plane
; i bit da 6 a 11 indicano quali planes sono abilitati alle collisioni
; i bit da 12 a 15 indicano quali degli sprite dispari sono abilitati
; al rilevamento delle collisioni.

                    ;5432109876543210
        dc.w        $98,%0000001111000111        ; CLXCON

; I planes 1,2,3,4 sono attivi per le collisioni (bit 6,7,8,9).
; viene segnalata collisione tra i playfield quando si sovrappongono
```

```
; un pixel che ha:        plane 1 = 1 (bit 0)
;                       plane 3 = 1 (bit 2)
; cioe‘ con il colore 3 del playfield 1
; e un pixel che ha:        plane 2 = 1 (bit 1)
;                       plane 4 = 0 (bit 3)
; cioe‘ con il colore 1 del playfield 2


        dc.w        $180    ; COLOR00
rileva_collisione:
        dc.w        0        ; IN QUESTO PUNTO la routine CheckColl modifica
                        ; la copper list scrivendo il colore giusto.

                                ; palette playfield 1
        dc.w        $182,$005       ; colori da 0 a 7
        dc.w        $184,$a40
        dc.w        $186,$f80
        dc.w        $188,$f00
        dc.w        $18a,$0f0
        dc.w        $18c,$00f
        dc.w        $18e,$080


                                ; palette playfield 2
        dc.w        $192,$367       ; colori da 9 a 15
        dc.w        $194,$0cc        ; il colore 8 e‘ trasparente, non va settato
        dc.w        $196,$a0a
        dc.w        $198,$242
        dc.w        $19a,$282
        dc.w        $19c,$861
        dc.w        $19e,$ff0

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

        dcb.b        40*90,0        ; questo spazio azzerato serve perche’ spostandoci
                        ; a visualizzare piu’ in basso e piu’ in alto usciamo
                        ; dalla zona della PIC1 e visualizziamo quello che sta
                        ; prima e dopo la pic stessa, il che’ causerebbe
                        ; la visualizzazione di byte sparsi di disturbo.
                        ; mettendo dei byte azzerati in quel punto viene
                        ; visualizzato $0000, ossia il colore di sfondo.

PIC1:       incbin        "colldual1.raw"
        dcb.b        40*30,0        ; vedi sopra

PIC2:       incbin        "colldual2.raw"

        end
```

In questo esempio mostriamo la collisione tra due playfield. Il meccanismo e‘ lo stesso delle collisioni tra gli sprite. Il registro CLXCON viene usato per indicare quali planes sono attivi per il rilevamento delle collisioni. Come al solito e‘ possibile indicare quali planes sono attivi, e quali valori devono assumere affinche‘ la collisione sia rilevata.
Nell’esempio rileviamo le collisioni tra colore 3 del playfield 1 e colore 1 del playfield 2. Se modificate la copperlist cambiando il valore di CLXCON, potete rivelare altri tipi di collisione. Ad esempio provate cosi‘:

```
        dc.w        $98,%0000001111000110        ; CLXCON
```

```
 I planes 1,2,3,4 sono attivi per le collisioni (bit 6,7,8,9).
 Viene segnalata collisione tra i playfield quando si sovrappongono
```

```
 un pixel che ha:        plane 1 = 0 (bit 0)
                               plane 3 = 1 (bit 2)
 cioe' il colore 2 del playfield 1
 e un pixel che ha:       plane 2 = 1 (bit 1)
                               plane 4 = 0 (bit 3)
 cioe' il colore 1 del playfield 2

Potete rilevare collisioni tra piu' colori non abilitando alcuni planes.
Esempio:

        dc.w        $98,%0000001011000011        ; CLXCON

 I planes 1,2 e 4 sono attivi per le collisioni (bit 6,7 e 9).
 Per quant riguarda il playfield 2 sono attivi entrambi i planes pertanto
 verranno considerati i pixel che hanno:        plane 2 = 1 (bit 1)
                                                       plane 4 = 0 (bit 3)
cioe' il colore 1 del playfield 2

Per quanto riguarda il playfield 1 invece e' abilitato solo il plane 1
e il valore del plane 3 non ha importanza.
Verranno considerati sia i pixel che hanno:        plane 0 = 1 (bit 0)
                                                       plane 3 = 0 (bit 2)
 sia i pixel che hanno:                          plane 0 = 1 (bit 0)
                                                       plane 3 = 1 (bit 2)

Cioe' vengono considerati sia il colore 1 del playfield 1, che il colore 3
del playfield 1.

Per il rilevamento vero e proprio si usa come al solito un bit di CLXDAT.
In questo caso si tratta del bit 0. Se vale 1 c'e' collisione tra i colori
specificati con CLXCON, altrimenti no.
```

23.23 Lezione7y

```
; Lezione7y1.s        UNO SPRITE VISUALIZZATO SCRIVENDO DIRETTAMENTE I REGISTRI
;                (SENZA DMA)
;       Questo esempio mostra uno sprite ottenuto usando direttamente i
;       registri. Lo sprite tra l'altro viene visualizzato a due diverse
;        posizioni orizzontali, in modo analogo al riutilizzo.


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
```

```
;        NON Puntiamo lo sprite !!!!!!!!!!!!!!!!!!!

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $1A2,$FF0        ; color17, ossia COLOR1 dello sprite0 - GIALLO
```

```
        dc.w        $1A4,$a00        ; color18, ossia COLOR2 dello sprite0 - ROSSO
        dc.w        $1A6,$F70        ; color19, ossia COLOR3 dello sprite0 - ARANCIO


        dc.w        $4007,$fffe       ; aspetta la linea $40
        dc.w        $140,$0080       ; SPR0POS - posizione orizzontale
        dc.w        $142,$0000       ; SPR0CTL
        dc.w        $146,$0e70       ; SPR0DATB
        dc.w        $144,$03c0       ; SPR0DATA - attiva lo sprite

        dc.w        $6007,$fffe       ; aspetta la linea $60
        dc.w        $142,$0000       ; SPR0CTL - "spegne" lo sprite

        dc.w        $140,$00a0       ; SPR0POS - nuova posizione orizz.
        dc.w        $146,$2ff4       ; SPR0DATB
        dc.w        $8007,$fffe       ; aspetta la linea $80
        dc.w        $144,$13c8       ; SPR0DATA - attiva lo sprite

        dc.w        $b407,$fffe       ; aspetta la linea $b4
        dc.w        $142,$0000       ; SPR0CTL - "spegne" lo sprite

        dc.w        $FFFF,$FFFE       ; Fine della copperlist



        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                       ; perche' per vedere gli sprite
                                       ; e' necessario che ci siano bitplanes
                                       ; abilitati
BITPLANE:
        ds.b        40*256               ; bitplane azzerato lowres

        end
```

In questo esempio vediamo come usare uno sprite manipolando direttamente i registri SPRxPOS, SPRxCTL, SPRxDATA, SPRxDATB.
Per prima cosa notate che NON puntiamo lo sprite nella copperlist. Anzi, per di piu‘ NON esiste la struttura sprite nella memoria chip. Infatti questa struttura e‘ usata dal DMA, che, quando e‘ usato,in pratica non fa altro che copiare i dati dalla struttura nei registri SPRxPOS, SPRxCTL, SPRxDATA, SPRxDATB. Se scriviamo noi i dati direttamente in quei registri, non abbiamo bisogno del DMA. Vediamo in dettaglio come si usano i registri.
In SPRxPOS, viene scritta la posizione dello sprite. Il contenuto di questo registro e‘ in pratica lo stesso della prima word di controllo della struttura sprite che usiamo con il DMA. La differenza, pero‘ e‘ che VSTART non influenza la posizione verticale degli sprite. Gli sprite infatti vengono attivati scrivendo nel registro SPRxDATA. Una volta attivato, uno sprite, viene disegnato ad ogni riga, alla posizione orizzontale che abbiamo scritto in SPRxPOS, e per ogni riga esso ha sempre la stessa "forma".
La "forma" dello sprite viene scritta nei due registri SPRxDATB e SPRxDATA che funzionano esattamente come le coppie di word che descrivono la forma dello sprite nella struttura che si usa con il DMA. I bit piu‘ significativi sono contenuti in SPRxDATB e i meno significativi in SPRxDATA. Questi due registri vengono riutilizzati per ogni riga. Se quindi si desidera che la forma dello sprite cambi da una riga ad un’altra e‘ necessario modificare i due registri SPRxDATx ad ogni riga.
Il registro SPRxCTL, invece ha lo stesso contenuto della seconda word di controllo della struttura. Anche qui la posizione verticale non serve a niente. In pratica, di tutto il registro gli unici bit che hanno un significato, sono il bit 0, che e‘ il bit basso di HSTART e il bit 7 che serve per "attaccare" gli sprite. Scrivendo nel registro SPRxCTL, inoltre, si disabilita lo sprite.

Usare gli sprite senza DMA e‘ molto scomodo per il fatto di dover cambiare ad ogni riga SPRxDATx. Infatti di solito non viene usato. Puo‘ diventare vantaggioso, pero‘ nel caso in cui si voglia uno sprite che sia uguale per ogni riga: in pratica per fare delle colonne. In questo caso infatti non e‘ necessario cambiare ad ogni riga SPRxDATx, perche‘ quello che si vuole e‘ appunto che ad ogni riga lo sprite abbia la stessa forma. Inoltre con questo metodo risparmiamo molta memoria: se dovessimo fare uno sprite-colonna alto 100 righe con il DMA, saremmo infatti costretti a utilizzare una struttura lunga 100 longword, escluse le word di controllo!

La procedura per fare una colonna con gli sprite senza DMA, quindi,
e‘ la seguente:
1) si scrivono i giusti valori in SPRxPOS, SPRxCTL e SPRxDATB
2) si attende la posizione verticale nella quale si vuole far iniziare lo sprite.
3) si scrive il valore di SPRxDATA. A questo punto lo sprite verra‘ disegnato, sempre uguale ad ogni riga.
4) si attende la posizione verticale nella quale si vuole far finire lo sprite.
5) si scrive un qualsiasi valore in SPRxCTL

E‘ possibile, come facciamo in questo esempio, visualizzare piu‘ colonne a diverse altezze, ripetendo la procedura precedentemente descritta piu‘ volte nella stessa copperlist. Si potrebbe anche cambiare la palette tra una colonna e l’altra.

Lezione7y2

```
; lezione7y2.s        barre verticali

;         In questo esempio utiliziamo 2 sprite per fare delle barre verticali.

        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        NON Puntiamo lo sprite !!!!!!!!!!!!!!!!!!!

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
```

```
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s         mouse

        bsr.s         MuoviSprite         ; Muove gli sprite 0 ed 1 orizzontalmente, ma
                                          ; agendo sui MOVE della COPPERLIST, dato
                                          ; che li visualiziamo tramite i registri
                                          ; diretti.

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s         Aspetta

        btst         #6,$bfe001           ; mouse premuto?
        bne.s         mouse

        move.l        OldCop(PC),$dff080   ; Puntiamo la cop di sistema
        move.w        d0,$dff088           ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr          -$7e(a6)             ; Enable
        move.l        gfxbase(PC),a1
        jsr          -$19e(a6)            ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0

OldCop:
        dc.l         0

; Questa routine sposta gli sprite agendo sulla copperlist, modificando
; il valore bar1 che viene caricato nel registro SPRxPOS, ossia
; il byte della  posizione X, immettendoci delle coordinate gia' stabilite
; nella tabella TABX

MuoviSprite:
        ADDQ.L        #2,TABXPOINT        ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in loong TABXPOINT
                                  ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTART        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima long
NOBSTART:
        MOVE.w        (A0),d1

        add.w        #128,D1               ; 128 - per centrare lo sprite.
        btst.l        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset.b        #0,bar1_b        ; Settiamo il bit basso della bar
        bra.s        PlaceCoords

BitBassoZERO:
        bclr.b        #0,bar1_b        ; Azzeriamo il bit basso della bar
PlaceCoords:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra

        move.b        D1,bar1               ; Poniamo il valore XX nel byte della posizione
```

```
        ADDQ.L        #2,TABXPOINT2         ; Fai puntare al byte successivo
        MOVE.L        TABXPOINT2(PC),A0 ; indirizzo contenuto in loong TABXPOINT
                              ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima longword della TAB?
        BNE.S        NOBSTART2        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT2 ; Riparti a puntare dalla prima long
NOBSTART2:
        MOVE.w        (A0),d1
        add.w        #128,D1               ; 128 - per centrare lo sprite.
        btst.l        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO2
        bset.b        #0,bar2_b        ; Settiamo il bit basso della bar
        bra.s        PlaceCoords2

BitBassoZERO2:
        bclr.b        #0,bar2_b        ; Azzeriamo il bit basso della bar
PlaceCoords2:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra

        move.b        D1,bar2                ; Poniamo il valore XX nel byte della posizione
        rts

TABXPOINT:
        dc.l        TABX-2

TABXPOINT2:                        ; il puntatore per il secondo sprite e' diverso
        dc.l        TABX+40-2

; Tabella con coordinate X dello sprite precalcolate.

TABX:
        incbin        "XCOORDINATOK.TAB"

FINETABX:

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
```

```
                                        ; in questo caso e' vuoto,
                                        ; per cui non compare.

        dc.w        $1A2,$FF0       ; color17, ossia COLOR1 dello sprite0 - GIALLO
        dc.w        $1A4,$a00       ; color18, ossia COLOR2 dello sprite0 - ROSSO
        dc.w        $1A6,$F70       ; color19, ossia COLOR3 dello sprite0 - ARANCIO


        dc.w        $2c07,$fffe       ; WAIT - aspetta il bordo superiore

        dc.w        $140              ; SPR0POS
        dc.b        0               ; posizione verticale (non usata)
bar1:       dc.b    0               ; posizione orizzontale
        dc.w        $142              ; SPR0CTL
        dc.b        0               ; VSTOP (non usato)
bar1_b:       dc.b        0               ; quarto byte di controllo: viene usato il
                                ; bit 0 che e' il bit basso della posizione
                                ; orizzontale

        dc.w        $146,$0e70        ; SPR0DATB
        dc.w        $144,$03c0        ; SPR0DATA - attiva lo sprite

        dc.w        $148              ; SPR1POS
        dc.b        0
bar2:       dc.b        0               ; posizione orizzontale
        dc.w        $14a              ; SPR1CTL
        dc.b        0
bar2_b:       dc.b        0
        dc.w        $14e,$3e7c        ; SPR1DATB
        dc.w        $14c,$0ff0;db0        ; SPR1DATA - attiva lo sprite


        dc.w        $FFFF,$FFFE       ; Fine della copperlist



        SECTION       PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256              ; bitplane azzerato lowres

        end
```

Notate che siccome le colonne sono alte tutto lo schermo, non e' necessario
scrivere nei registri SPRxCTL per disabilitare gli sprite.

Inserite questo pezzo di copperist appena prima del "dc.w $FFFF,$FFFE" per
dare un tocco di colore al listato. (Amiga+b+c+i)

```
        dc.w        $5407,$fffe       ; WAIT
        dc.w        $1A2,$FaF       ; color17       ; tono viola
        dc.w        $1A4,$703       ; color18
        dc.w        $1A6,$F0a       ; color19
        dc.w        $6807,$fffe       ; WAIT
        dc.w        $1A2,$aFa       ; color17       ; tono verde
        dc.w        $1A4,$050       ; color18
        dc.w        $1A6,$0a0       ; color19
        dc.w        $7c07,$fffe       ; WAIT
        dc.w        $1A2,$0FF       ; color17       ; tono blu
```

```
        dc.w        $1A4,$00d        ; color18
        dc.w        $1A6,$07F        ; color19
        dc.w        $9007,$fffe        ; WAIT
        dc.w        $1A2,$eee        ; color17        ; tono grigio
        dc.w        $1A4,$444        ; color18
        dc.w        $1A6,$888        ; color19
```

Lezione7y3

```
; Lezione7y3.s        DUE UTILIZZI DI UNO SPRITE SULLA STESSA RIGA

;       Questo esempio mostra come sia possibile riutilizzare uno sprite
;       2 volte su una stessa riga accedendo direttamente ai registri


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        NON Puntiamo lo sprite !!!!!!!!!!!!!!!!!!!

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
```

```
        dc.l        0

OldCop:
        dc.l        0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                              ; in questo caso e' vuoto,
                                              ; per cui non compare.

        dc.w        $1A2,$FF0        ; color17, ossia COLOR1 dello sprite0 - GIALLO
        dc.w        $1A4,$a00        ; color18, ossia COLOR2 dello sprite0 - ROSSO
        dc.w        $1A6,$F70        ; color19, ossia COLOR3 dello sprite0 - ARANCIO

; ---> inserite qua il pezzo di copperlist riportato in fondo al commento

        dc.w        $4007,$fffe        ; aspetta la linea $40, posizione orizz. 7
        dc.w        $140,$0060        ; SPR0POS - posizione orizzontale
        dc.w        $142,$0000        ; SPR0CTL
        dc.w        $146,$0e70        ; SPR0DATB
        dc.w        $144,$03c0        ; SPR0DATA - attiva lo sprite

        dc.w        $4087,$fffe        ; aspetta la linea $40, posizione orizz. 87
        dc.w        $140,$00a0        ; SPR0POS - posizione orizzontale

; lo stesso per la linea $41
        dc.w        $4107,$fffe        ; wait pos. orizzontale 07
        dc.w        $140,$0060        ; spr0pos

        dc.w        $4187,$fffe        ; wait pos. orizzontale 87
        dc.w        $140,$00a0        ; spr0pos

; lo stesso per la linea $42
        dc.w        $4207,$fffe        ; wait
        dc.w        $140,$0060        ; spr0pos

        dc.w        $4287,$fffe        ; wait... eccetera
```

```
        dc.w        $140,$00a0

; lo stesso per la linea $43
        dc.w        $4307,$fffe
        dc.w        $140,$0060

        dc.w        $4387,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $44
        dc.w        $4407,$fffe
        dc.w        $140,$0060

        dc.w        $4487,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $45
        dc.w        $4507,$fffe
        dc.w        $140,$0060

        dc.w        $4587,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $46
        dc.w        $4607,$fffe
        dc.w        $140,$0060

        dc.w        $4687,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $47
        dc.w        $4707,$fffe
        dc.w        $140,$0060

        dc.w        $4787,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $48
        dc.w        $4807,$fffe
        dc.w        $140,$0060

        dc.w        $4887,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $49
        dc.w        $4907,$fffe
        dc.w        $140,$0060

        dc.w        $4987,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $4a
        dc.w        $4a07,$fffe
        dc.w        $140,$0060

        dc.w        $4a87,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $4b
        dc.w        $4b07,$fffe
        dc.w        $140,$0060
```

```
        dc.w        $4b87,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $4c
        dc.w        $4c07,$fffe
        dc.w        $140,$0060

        dc.w        $4c87,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $4d
        dc.w        $4d07,$fffe
        dc.w        $140,$0060

        dc.w        $4d87,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $4e
        dc.w        $4e07,$fffe
        dc.w        $140,$0060

        dc.w        $4e87,$fffe
        dc.w        $140,$00a0

; lo stesso per la linea $4f
        dc.w        $4f07,$fffe
        dc.w        $140,$0060

        dc.w        $4f87,$fffe
        dc.w        $140,$00a0

        dc.w        $5007,$fffe        ; aspetta la linea $50
        dc.w        $142,$0000        ; SPR0CTL - "spegne" lo sprite

        dc.w        $FFFF,$FFFE        ; Fine della copperlist



        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

Manipolare direttamente i registri rende anche possibile disegnare uno sprite due volte sulla stessa riga, ovvero disegnarlo in due diverse posizioni orizzontali. Il trucco fa uso del copper e della sua capacita‘ di aspettare che il pennello elettronico abbia raggiunto una determinata posizione sul video. All’inizio si aspetta con il copper la prima riga dello schermo in cui si vuole disegnare lo sprite. Nell’esempio aspettiamo la riga $40 mettendo nella copperlist:

```
        dc.w        $4007,$fffe        ; aspetta la linea $40, posizione orizz. 7
```

Poi si caricano i registri SPR0CTL, SPRDATB e SPRDAT:

```
        dc.w        $142,$0000        ; SPR0CTL
        dc.w        $146,$0e70        ; SPR0DATB
        dc.w        $144,$03c0        ; SPR0DATA - attiva lo sprite
```

E si mette il primo valore della posizione orizzontale in SPRxPOS:

```
        dc.w        $140,$0060        ; SPR0POS - posizione orizzontale
```

A questo punto si aspetta che il pennello elettronico superi questa posizione orizzontale, in modo che lo sprite venga disegnato.

```
        dc.w        $4087,$fffe        ; aspetta la linea $40, posizione orizz. 87
```

Nell'esempio la posizione orizzontale dello sprite e' $60. Aspettando la posizione $87 siamo abbondantemente sicuri che lo sprite e' stato disegnato. Infatti quando il pennello elettronico ha superato la posizione orizzontale, lo sprite e' stato effettivamente disegnato.

Una volta avvenuto cio', si scrive la seconda posizione orizzontale in SPRxPOS.

```
        dc.w        $140,$00a0        ; SPR0POS - posizione orizzontale
```

In questo modo lo sprite verra' disegnato anche nella seconda posizione orizzontale. A questo punto abbiamo disegnato 2 volte lo stesso sprite su una riga. Per disegnare i 2 sprite sulle righe seguenti, e' sufficente ripetere tutti i passi fin qui descritti.
Per esempio per la riga $41 scriviamo

```
; lo stesso per la linea $41
        dc.w        $4107,$fffe
        dc.w        $140,$0060

        dc.w        $4187,$fffe
        dc.w        $140,$00a0
```

che e' la stessa cosa della riga $40, solo che teniamo costanti SPRODATA SPR0DATB e SPR0CTL in modo da tenere costante la forma dello sprite.
Volendo si possono variare questi registri per cambiare la forma dello sprite tra una riga e l'altra.

Per disabilitare gli sprite basta scrivere un qualsiasi valore in SPL0CTL, in questo modo:

```
        dc.w        $5007,$fffe        ; aspetta la linea $50
        dc.w        $142,$0000        ; SPR0CTL - "spegne" lo sprite
```

Se proprio voleste esagerare, potete anche cambiare la palette orizzontalmente tra la prima barra e la seconda, in modo da avere riutilizzi sulla stessa linea anche di colore diverso. Provate ad inserire questa copperlist nel punto indicato con:

```
; ---> inserite qua il pezzo di copperlist riportato in fondo al commento
```

Nella copperlist del listato. In pratica sostituiamo la parte finale che si occupa di visualizzare lo sprite. (Amiga+b+c+i per copiare il testo)

```
        dc.w        $4007,$fffe        ; aspetta la linea $40, posizione orizz. 7
        dc.w        $140,$0060        ; SPR0POS - posizione orizzontale
        dc.w        $142,$0000        ; SPR0CTL
        dc.w        $146,$0e70        ; SPR0DATB
        dc.w        $144,$03c0        ; SPR0DATA - attiva lo sprite
```

```
        dc.w        $4087,$fffe        ; aspetta la linea $40, posizione orizz. 87
        dc.w        $1A2,$aFa        ; color17        ; tono verde
        dc.w        $1A4,$050        ; color18
        dc.w        $1A6,$0a0        ; color19
        dc.w        $140,$00a0        ; SPR0POS - posizione orizzontale

; linea $41
        dc.w        $4107,$fffe        ; wait pos. orizzontale 07
        dc.w        $1A2,$FF0        ; color17        ; tono arancio
        dc.w        $1A4,$a00        ; color18
        dc.w        $1A6,$F70        ; color19
        dc.w        $140,$0060        ; spr0pos
        dc.w        $4187,$fffe        ; wait pos. orizzontale 87
        dc.w        $1A2,$aFa        ; color17        ; tono verde
        dc.w        $1A4,$050        ; color18
        dc.w        $1A6,$0a0        ; color19
        dc.w        $140,$00a0        ; spr0pos
; linea $42
        dc.w        $4207,$fffe        ; wait pos. orizzontale 07
        dc.w        $1A2,$FF0        ; color17        ; tono arancio
        dc.w        $1A4,$a00        ; color18
        dc.w        $1A6,$F70        ; color19
        dc.w        $140,$0060        ; spr0pos
        dc.w        $4287,$fffe        ; wait pos. orizzontale 87
        dc.w        $1A2,$aFa        ; color17        ; tono verde
        dc.w        $1A4,$050        ; color18
        dc.w        $1A6,$0a0        ; color19
        dc.w        $140,$00a0        ; spr0pos
; linea $43
        dc.w        $4307,$fffe        ; wait pos. orizzontale 07
        dc.w        $1A2,$FF0        ; color17        ; tono arancio
        dc.w        $1A4,$a00        ; color18
        dc.w        $1A6,$F70        ; color19
        dc.w        $140,$0060        ; spr0pos
        dc.w        $4387,$fffe        ; wait pos. orizzontale 87
        dc.w        $1A2,$aFa        ; color17        ; tono verde
        dc.w        $1A4,$050        ; color18
        dc.w        $1A6,$0a0        ; color19
        dc.w        $140,$00a0        ; spr0pos
; linea $44
        dc.w        $4407,$fffe        ; wait pos. orizzontale 07
        dc.w        $1A2,$FF0        ; color17        ; tono arancio
        dc.w        $1A4,$a00        ; color18
        dc.w        $1A6,$F70        ; color19
        dc.w        $140,$0060        ; spr0pos
        dc.w        $4487,$fffe        ; wait pos. orizzontale 87
        dc.w        $1A2,$aFa        ; color17        ; tono verde
        dc.w        $1A4,$050        ; color18
        dc.w        $1A6,$0a0        ; color19
        dc.w        $140,$00a0        ; spr0pos
; linea $45
        dc.w        $4507,$fffe        ; wait pos. orizzontale 07
        dc.w        $1A2,$FF0        ; color17        ; tono arancio
        dc.w        $1A4,$a00        ; color18
        dc.w        $1A6,$F70        ; color19
        dc.w        $140,$0060        ; spr0pos
        dc.w        $4587,$fffe        ; wait pos. orizzontale 87
        dc.w        $1A2,$aFa        ; color17        ; tono verde
        dc.w        $1A4,$050        ; color18
        dc.w        $1A6,$0a0        ; color19
        dc.w        $140,$00a0        ; spr0pos
```

```
        dc.w        $4607,$fffe        ; aspetta la linea $46
        dc.w        $142,$0000        ; SPR0CTL - "spegne" lo sprite
        dc.w        $ffff,$fffe        ; fine copperlist
```

Lezione7y4

```
; Lezione7y4.s        - Paesaggio fatto con 2 soli sprite!!

;       Questo esempio mostra come sia possibile generare una intera
;        schermata usando direttamente i registri degli sprite


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        NON Puntiamo lo sprite !!!!!!!!!!!!!!!!!!!


        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0
```

```
OldCop:
        dc.l        0


        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $01ba,$0fff                ; colore 29
        dc.w        $01bc,$0aaa                ; colore 30
        dc.w        $01be,$0753                ; colore 31

; per comodita' usiamo dei simboli. Ricordo che si puo' definire un simbolo
; o EQUATE in due modi, e cioe', come in questo caso, ponendo il nome del
; simbolo che vogliamo creare senza spaziature, seguito da un = e dal valore
; che tale simbolo dovra' rappresentare, oppure nella stesso modo, ma con il
; simbolo EQU anziche' l'=.

spr6pos                = $170
spr6data        = $174
spr6datb        = $176
spr7pos                = $178
spr7data        = $17c
spr7datb        = $17e

; linea $50
        dc.w        $5025,$fffe
        dc.w        spr6data,$0,spr6datb,$0,spr7data,$f000,spr7datb,$0
        dc.w        spr6pos,$40,spr7pos,$48,$504b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$505b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$506b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$507b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$508b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$509b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$50ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$50bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$50cb,$fffe
```

```
        dc.w        spr6pos,$d0,spr7pos,$d8,$50db,$fffe

; linea $51
        dc.w        $5125,$fffe
        dc.w        spr6data,$0001,spr6datb,$0000,spr7data,$b800,spr7datb,$4000
        dc.w        spr6pos,$40,spr7pos,$48,$514b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$515b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$516b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$517b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$518b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$519b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$51ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$51bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$51cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$51db,$fffe

; linea $52
        dc.w        $5225,$fffe
        dc.w        spr6data,$0003,spr6datb,$0000,spr7data,$bc00,spr7datb,$4000
        dc.w        spr6pos,$40,spr7pos,$48,$524b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$525b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$526b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$527b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$528b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$529b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$52ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$52bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$52cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$52db,$fffe

; linea $53
        dc.w        $5325,$fffe
        dc.w        spr6data,$0002,spr6datb,$0001,spr7data,$ec00,spr7datb,$1200
        dc.w        spr6pos,$40,spr7pos,$48,$534b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$535b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$536b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$537b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$538b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$539b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$53ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$53bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$53cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$53db,$fffe

; linea $54
        dc.w        $5425,$fffe
        dc.w        spr6data,$0007,spr6datb,$0000,spr7data,$2b00,spr7datb,$d400
        dc.w        spr6pos,$40,spr7pos,$48,$544b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$545b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$546b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$547b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$548b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$549b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$54ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$54bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$54cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$54db,$fffe

; linea $55
        dc.w        $5525,$fffe
        dc.w        spr6data,$001c,spr6datb,$0003,spr7data,$e780,spr7datb,$1800
        dc.w        spr6pos,$40,spr7pos,$48,$554b,$fffe
```

```
        dc.w        spr6pos,$50,spr7pos,$58,$555b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$556b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$557b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$558b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$559b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$55ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$55bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$55cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$55db,$fffe

; linea $56
        dc.w        $5625,$fffe
        dc.w        spr6data,$803e,spr6datb,$0001,spr7data,$9ac1,spr7datb,$6500
        dc.w        spr6pos,$40,spr7pos,$48,$564b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$565b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$566b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$567b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$568b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$569b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$56ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$56bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$56cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$56db,$fffe

; linea $57
        dc.w        $5725,$fffe
        dc.w        spr6data,$c079,spr6datb,$0006,spr7data,$b6e7,spr7datb,$4910
        dc.w        spr6pos,$40,spr7pos,$48,$574b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$575b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$576b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$577b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$578b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$579b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$57ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$57bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$57cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$57db,$fffe

; linea $58
        dc.w        $5825,$fffe
        dc.w        spr6data,$c07f,spr6datb,$0048,spr7data,$fff6,spr7datb,$2009
        dc.w        spr6pos,$40,spr7pos,$48,$584b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$585b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$586b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$587b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$588b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$589b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$58ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$58bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$58cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$58db,$fffe

; linea $59
        dc.w        $5925,$fffe
        dc.w        spr6data,$e06f,spr6datb,$0096,spr7data,$7eaf,spr7datb,$a150
        dc.w        spr6pos,$40,spr7pos,$48,$594b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$595b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$596b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$597b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$598b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$599b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$59ab,$fffe
```

```
        dc.w        spr6pos,$b0,spr7pos,$b8,$59bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$59cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$59db,$fffe

; linea $5a
        dc.w        $5a25,$fffe
        dc.w        spr6data,$61ed,spr6datb,$9013,spr7data,$dfff,spr7datb,$6cab
        dc.w        spr6pos,$40,spr7pos,$48,$5a4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5a5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5a6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5a7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5a8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5a9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5aab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5abb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5acb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5adb,$fffe

; linea $5b
        dc.w        $5b25,$fffe
        dc.w        spr6data,$db9f,spr6datb,$72ed,spr7data,$ffff,spr7datb,$dbee
        dc.w        spr6pos,$40,spr7pos,$48,$5b4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5b5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5b6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5b7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5b8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5b9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5bab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5bbb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5bcb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5bdb,$fffe

; linea $5c
        dc.w        $5c25,$fffe
        dc.w        spr6data,$ffff,spr6datb,$cfbf,spr7data,$ffff,spr7datb,$ff3f
        dc.w        spr6pos,$40,spr7pos,$48,$5c4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5c5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5c6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5c7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5c8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5c9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5cab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5cbb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5ccb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5cdb,$fffe

; linea $5d
        dc.w        $5d25,$fffe
        dc.w        spr6data,$ffff,spr6datb,$ffff,spr7data,$ffff,spr7datb,$feff
        dc.w        spr6pos,$40,spr7pos,$48,$5d4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5d5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5d6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5d7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5d8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5d9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5dab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5dbb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5dcb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5ddb,$fffe

; istruzioni copper per disattivare gli sprite
        dc.w        $5107,$fffe                ; aspetta inizio riga
```

```
        dc.w        $172,0                      ; spr6ctl
        dc.w        $17a,0                      ; spr7ctl

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                       ; perche' per vedere gli sprite
                                       ; e' necessario che ci siano bitplanes
                                       ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

Come vedete e' possibile visualizzare anche piu' di 2 volte uno sprite sulla stessa riga, a patto che si sappia programmare in assembler.
In questo esempio usiamo 2 sprite (6 e 7) visualizzandoli 10 volte ciascuno per linea, per un totale di 16*20=320 pixel per riga.
In pratica copriamo tutto lo schermo. L'idea e' la stessa dell'esempio precedente, cioe' di cambiare i valori dei registri degli sprite con il copper. Questa volta pero' oltre a cambiare la posizione cambiamo ad ogni riga anche la forma degli sprite modificando i valori dei registri SPRxDATA e SPRxDATB, in modo da formare un paesaggio. Per semplicita' il nostro paesaggio e' alto 14 righe, ma si potrebbe riempire lo schermo!
Ogni linea della copperlist e' fatta in questo modo:

```
; linea $50
        dc.w        $5025,$fffe
        dc.w        spr6data,$0,spr6datb,$0,spr7data,$f000,spr7datb,$0
        dc.w        spr6pos,$40,spr7pos,$48,$504b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$505b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$506b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$507b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$508b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$509b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$50ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$50bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$50cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$50db,$fffe
```

In questo caso abbiamo preso la parte di copperlist relativa alla riga $50.
Vediamo il significato di tutte le istruzioni:

```
        dc.w        $5025,$fffe        ; WAIT
```

La prima istruzione serve ad attendere che il pennello elettronico raggiunga la posizione orizzontale $25 della riga attuale.

```
        dc.w        spr6data,$0,spr6datb,$0,spr7data,$f000,spr7datb,$0
```

Queste istruzioni sevono a settare per questa riga i valori dei registri DATA che determinano la forma dello sprite.

```
        dc.w        spr6pos,$40,spr7pos,$48,$504b,$fffe
```

queste istruzioni servono per cambiare le posizioni degli sprite.
Lavorano come nell'esempio precedente. Prima vengono aggiornate le posizioni degli sprite, e poi si attende che i 2 sprite vengano visualizzati.
A questo punto si ripetono 10 gruppi di istruzioni copper come questo, che a loro volta modificano le posizioni degli sprite e attendono con dei WAIT che distanziano di 16 pixel orizzontali ($4b, $5b, $6b...) gli sprite visualzzati.

```
Per es. troviamo

        dc.w        spr6pos,$50,spr7pos,$58,$505b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$506b,$fffe

Dividiamo una linea nei 3 comandi che contiene:

        dc.w        spr6pos,$50        ; determina la posizione dello sprite6
        dc.w        spr7pos,$58        ; determina la posizione dello sprite7
        dc.w        $505b,$fffe        ; WAIT - attendi 16 pixel piu' avanti.

e cosi‘ via.
Dopo 10 gruppi cosi‘ abbiamo disegnato tutta una riga. A questo punto non ci
resta che ripetere tutto quello che abbiamo fatto per la riga $50 anche per
tutte le altre righe del paesaggio. Ovviamente per ogni riga avremo un
diverso valore nei registri SPRxDATx che determinera‘ una diversa forma per
lo sprite.

Naturalmente per generare copperlist cosi' lunghe e complesse vengono scritte
apposite routine "GeneraCopperlist", che pero' per la loro complessita' non
sono ancora state inserite nel corso. L'importante in questo listato e' capire
il meccanismo del riutilizzo degli sprite agendo direttamente sui registri con
la copperlist.
```

Lezione7y5

```
; Lezione7y5.s        - Paesaggio fatto con 2 soli sprite che scorre

;       Questo esempio mostra come sia possibile generare una intera
;        schermata usando direttamente i registri di 2 sprite (il 6 e il 7)
;       La schermata viene inoltre "scrollata"


        SECTION        CiriCop,CODE

Inizio:
        move.l        4.w,a6                ; Execbase
        jsr        -$78(a6)        ; Disable
        lea        GfxName(PC),a1        ; Nome lib
        jsr        -$198(a6)        ; OpenLibrary
        move.l        d0,GfxBase
        move.l        d0,a6
        move.l        $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo un bitplane "vuoto"

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        NON Puntiamo lo sprite !!!!!!!!!!!!!!!!!!!


        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!
```

```
mouse:
        cmpi.b        #$aa,$dff006        ; Linea $aa?
        bne.s         mouse

        bsr.w         MuoviPaesaggio        ; Fa scrorrere il paesaggio

Aspetta:
        cmpi.b        #$aa,$dff006        ; linea $aa?
        beq.s         Aspetta

        btst         #6,$bfe001        ; mouse premuto?
        bne.s         mouse


        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr         -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr         -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b         "graphics.library",0,0

GfxBase:
        dc.l         0

OldCop:
        dc.l         0


; Questa routine fa scorrere i dati degli sprite che formano il paesaggio

MuoviPaesaggio:
        moveq         #14-1,d0        ; numero di righe
        lea         FormaSprite,a0        ; indirizzo primi dati dello sprite
PaeLoop:

; fa scorrere il piano A degli sprite

        move.w         (a0),d1                ; legge valore di spr6data
        swap         d1                ; lo mette nella word alta del registro
        move.w         8(a0),d1        ; legge valore di spr7data

        ror.l         #1,d1                ; fa scorrere i bit della forma degli sprite
        move.w         d1,8(a0)        ; scrive valore di spr7data
        swap         d1                ; scambia le word del registro
        move.w         d1,(a0)                ; scrive valore di spr6data

; fa scorrere il piano B degli sprite

        move.w         4(a0),d1        ; legge valore di spr6datb
        swap         d1                ; lo mette nella word alta del registro
        move.w         12(a0),d1         ; legge valore di spr7datb

        ror.l         #1,d1                ; fa scorrere i bit della forma degli sprite
        move.w         d1,12(a0)        ; scrive valore di spr7datb
```

```
        swap        d1                ; scambia le word del registro
        move.w        d1,4(a0)        ; scrive valore di spr6datb

        add.w        #140,a0                ; prossima riga del paesaggio
        dbra        d0,PaeLoop

        rts



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w        $01ba,$0fff                ; colore 29
        dc.w        $01bc,$0aaa                ; colore 30
        dc.w        $01be,$0753                ; colore 31

; per comodita‘ usiamo dei simboli

spr6pos                = $170
spr6data        = $174
spr6datb        = $176
spr7pos                = $178
spr7data        = $17c
spr7datb        = $17e

; linea $50 - (le istruzioni copper per una linea sono lunghe 140 bytes)

        dc.w        $5025,$fffe        ; Wait
        dc.w        spr6data
FormaSprite:                        ; Da questa label faremo gli offset per
        dc.w        $0                ; raggiungere tutti gli altri sprxdat
        dc.w        spr6datb
        dc.w        $0
        dc.w        spr7data
        dc.w        $f000
```

```
        dc.w        spr7datb
        dc.w        $0
        dc.w        spr6pos,$40,spr7pos,$48,$504b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$505b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$506b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$507b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$508b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$509b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$50ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$50bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$50cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$50db,$fffe

; linea $51
        dc.w        $5125,$fffe
        dc.w        spr6data,$0001,spr6datb,$0000,spr7data,$b800,spr7datb,$4000
        dc.w        spr6pos,$40,spr7pos,$48,$514b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$515b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$516b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$517b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$518b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$519b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$51ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$51bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$51cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$51db,$fffe

; linea $52
        dc.w        $5225,$fffe
        dc.w        spr6data,$0003,spr6datb,$0000,spr7data,$bc00,spr7datb,$4000
        dc.w        spr6pos,$40,spr7pos,$48,$524b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$525b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$526b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$527b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$528b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$529b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$52ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$52bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$52cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$52db,$fffe

; linea $53
        dc.w        $5325,$fffe
        dc.w        spr6data,$0002,spr6datb,$0001,spr7data,$ec00,spr7datb,$1200
        dc.w        spr6pos,$40,spr7pos,$48,$534b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$535b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$536b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$537b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$538b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$539b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$53ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$53bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$53cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$53db,$fffe

; linea $54
        dc.w        $5425,$fffe
        dc.w        spr6data,$0007,spr6datb,$0000,spr7data,$2b00,spr7datb,$d400
        dc.w        spr6pos,$40,spr7pos,$48,$544b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$545b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$546b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$547b,$fffe
```

```
        dc.w        spr6pos,$80,spr7pos,$88,$548b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$549b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$54ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$54bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$54cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$54db,$fffe

; linea $55
        dc.w        $5525,$fffe
        dc.w        spr6data,$001c,spr6datb,$0003,spr7data,$e780,spr7datb,$1800
        dc.w        spr6pos,$40,spr7pos,$48,$554b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$555b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$556b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$557b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$558b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$559b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$55ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$55bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$55cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$55db,$fffe

; linea $56
        dc.w        $5625,$fffe
        dc.w        spr6data,$803e,spr6datb,$0001,spr7data,$9ac1,spr7datb,$6500
        dc.w        spr6pos,$40,spr7pos,$48,$564b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$565b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$566b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$567b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$568b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$569b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$56ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$56bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$56cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$56db,$fffe

; linea $57
        dc.w        $5725,$fffe
        dc.w        spr6data,$c079,spr6datb,$0006,spr7data,$b6e7,spr7datb,$4910
        dc.w        spr6pos,$40,spr7pos,$48,$574b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$575b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$576b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$577b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$578b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$579b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$57ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$57bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$57cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$57db,$fffe

; linea $58
        dc.w        $5825,$fffe
        dc.w        spr6data,$c07f,spr6datb,$0048,spr7data,$fff6,spr7datb,$2009
        dc.w        spr6pos,$40,spr7pos,$48,$584b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$585b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$586b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$587b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$588b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$589b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$58ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$58bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$58cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$58db,$fffe
```

```
; linea $59
        dc.w        $5925,$fffe
        dc.w        spr6data,$e06f,spr6datb,$0096,spr7data,$7eaf,spr7datb,$a150
        dc.w        spr6pos,$40,spr7pos,$48,$594b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$595b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$596b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$597b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$598b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$599b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$59ab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$59bb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$59cb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$59db,$fffe

; linea $5a
        dc.w        $5a25,$fffe
        dc.w        spr6data,$61ed,spr6datb,$9013,spr7data,$dfff,spr7datb,$6cab
        dc.w        spr6pos,$40,spr7pos,$48,$5a4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5a5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5a6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5a7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5a8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5a9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5aab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5abb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5acb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5adb,$fffe

; linea $5b
        dc.w        $5b25,$fffe
        dc.w        spr6data,$db9f,spr6datb,$72ed,spr7data,$ffff,spr7datb,$dbee
        dc.w        spr6pos,$40,spr7pos,$48,$5b4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5b5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5b6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5b7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5b8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5b9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5bab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5bbb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5bcb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5bdb,$fffe

; linea $5c
        dc.w        $5c25,$fffe
        dc.w        spr6data,$ffff,spr6datb,$cfbf,spr7data,$ffff,spr7datb,$ff3f
        dc.w        spr6pos,$40,spr7pos,$48,$5c4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5c5b,$fffe
        dc.w        spr6pos,$60,spr7pos,$68,$5c6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5c7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5c8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5c9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5cab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5cbb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5ccb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5cdb,$fffe

; linea $5d
        dc.w        $5d25,$fffe
        dc.w        spr6data,$ffff,spr6datb,$ffff,spr7data,$ffff,spr7datb,$feff
        dc.w        spr6pos,$40,spr7pos,$48,$5d4b,$fffe
        dc.w        spr6pos,$50,spr7pos,$58,$5d5b,$fffe
```

```
        dc.w        spr6pos,$60,spr7pos,$68,$5d6b,$fffe
        dc.w        spr6pos,$70,spr7pos,$78,$5d7b,$fffe
        dc.w        spr6pos,$80,spr7pos,$88,$5d8b,$fffe
        dc.w        spr6pos,$90,spr7pos,$98,$5d9b,$fffe
        dc.w        spr6pos,$a0,spr7pos,$a8,$5dab,$fffe
        dc.w        spr6pos,$b0,spr7pos,$b8,$5dbb,$fffe
        dc.w        spr6pos,$c0,spr7pos,$c8,$5dcb,$fffe
        dc.w        spr6pos,$d0,spr7pos,$d8,$5ddb,$fffe

; istruzioni copper per disattivare gli sprite

        dc.w        $5e07,$fffe                ; aspetta inizio riga
        dc.w        $172,0                        ; spr6ctl
        dc.w        $17a,0                        ; spr7ctl

        dc.w        $FFFF,$FFFE        ; Fine della copperlist



        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres

        end
```

In questo esempio facciamo scorrere il paesaggio fatto con gli sprite.
A tale scopo non e‘ possibile usare il registro BPLCON1, che abbiamo usato
per far scorrere i bit-planes, perche‘ esso non ha effetto sugli sprite.
Per far scorrere il paesaggio dobbiamo far scorrere tutti i pixel che lo
compongono. In questo caso ci e‘ molto utile il fatto che il paesaggio
sia composto sempre dalla stessa figura che si ripete orizzontalmente
ogni 32 pixel.
Il paesaggio infatti e‘ costituito da 2 sprite (16 pixel ognuno) che si
ripetono sempre uguali per tutta la riga. Per far scorrere tutto il paesaggio,
bastera‘ quindi far scorrere i pixel che compongono la forma dei 2 sprite.
Ma tali pixel vengono scritti ad ogni nuova riga del paesaggio nei registri
SPR6DATA, SPR6DATB, SPR7DATA e SPR7DATB.
Quindi dovremo far scorrere il contenuto di questi registri
che si trova nella copperlist. All’indirizzo FormaSprite nella copperlist
possiamo trovare il dato che viene scritto in SPR6DATA nella prima riga del
paesaggio. Rispettivamente 4,8 e 12 bytes dopo, c’e‘ il contenuto dei registri
SPR6DATB, SPR7DATA e SPR7DATB, sempre della prima riga.
Facendo scorrere il contenuto dei nostri registri in tutte le 14 righe del
paesaggio avremo raggiunto il nostro scopo.

Vediamo ora come si effettua lo scorrimento, per esempio verso destra

Noi conosciamo gia‘ un istruzione che fa scorrere i pixel di una locazione di
memoria o di un registro del 68000, la LSR. Questa istruzione fa scorrere
i bit verso destra e fa entrare a sinistra dei bit di valore 0.
Per esempio:

```
        move.b  #%00100101,d0
        lsr.b   #3,d0
```

dopo queste istruzioni in d0 ci sara‘ il valore %00000100
Nel nostro caso questo non va bene, perche‘ i bit di valore 0 che entrano da
sinistra producono un "buco" nello sprite che si allarga sempre di piu‘, fino

```
a cancellare del tutto la figura.

Quello che ci occorre e‘ un’istruzione che faccia scorrere i bit verso
destra, ma che faccia rientrare a sinistra i bit che sono usciti a destra.
Cioe‘ in pratica che faccia ruotare i bit all’interno del registro.
Questa istruzione esiste, e si chiama ROR ("ROtate Right")
Vediamo subito un esempio:

        move.b  #%00100101,d0
        ror.b   #3,d0

dopo queste istruzioni in d0 ci sara‘ il valore %10100100. In pratica i 3
bit che prima della ROR si trovavano a destra sono rientrati da sinistra,
seguendo il percorso indicato sotto:

bit:     7 6 5 4 3 2 1 0
        -----------------
        | | | | | | | | |
        -----------------
       ->   -->   -->   -->
      |                     |   direzione di scorrimento.
        <--   <--   <--   <-

La routine "MuoviPaesaggio" utilizza un’istruzione ROR per far scorrere
i pixel. Notate che il contenuto del registro SPR6DATA e il contenuto di
SPR7DATA vengono messi nello stesso registro e vengono fatti ruotare
insieme, in modo che i bit che escono a sinistra da SPR6DATA entrano a destra
in SPR7DATA e i bit che escono a sinistra da SPR7DATA entrano a destra in
SPR6DATA.
La stessa operazione viene ripetuta anche per i contenuti dei registri
SPR6DATB e SPR7DATB che costituiscono il secondo piano degli sprite.

Per rendervi conto meglio della differenza tra la LSR e la ROR, provate
a sostituire LSR al posto di ROR nella routine "MuoviPaesaggio".
Vedrete che non e‘ esattamente quello che volevamo!

Naturalmente oltre alla ROR esiste un’istruzione per ruotare i bit verso
sinistra, che si chiama ROL ("ROtate Left") e che funziona esattamente nello
stesso modo. Potete tranquillamente sostituirla a ROR e vedrete il paesaggio
scorrere in direzione opposta.

In teoria con soli 2 sprite si potrebbe riempire tutto lo schermo, partendo
dalle nuvole in alto, poi i monti, la prateria e gli alberelli in primo piano,
e si potrebbe anche far scorrere a diverse velocita’ i vari livelli della
parallasse, cioe’ i monti "lontani" piu’ piano, la prateria velocita’ media,
gli alberi in primo piano velocemente. Tutto cio’ richiederebbe una copperlist
enorme e la routine per muovere tutto sarebbe molto lenta, questo e’ uno dei
motivi per cui le parallassi di questo genere sono fatte con i bitplanes, che
sono piu’ colorati e veloci da muovere. Comunque se qualcuno osasse dire che
l’Amiga ha solo 8 piccoli sprites, potreste fargli vedere la schermata in
parallasse fatta con gli sprites in sovraimpressione su un disegno a 4096
colori HAM, e vi rimarrebbero anche 6 sprites per farci le stelline e qualche
astronave. Se poi ci faceste girare in mezzo anche un centinaio di BOB col
blitter... forse non si capirebbe piu’ niente, ma sarebbe interessante.
```

23.24 Lezione7z

```
; Lezione7z.s        ANIMAZIONE (6 FOTOGRAMMI) DI UNO SPRITE ATTACCHED
```

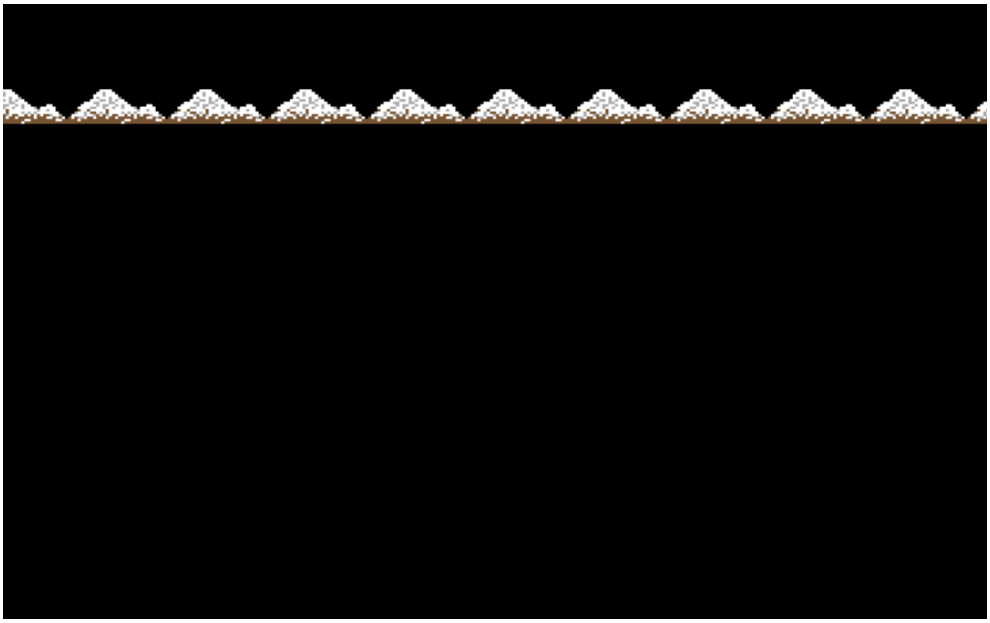


Figura 23.11: Lezione 7y5

```
        SECTION        CiriCop,CODE

Inizio:
        move.l         4.w,a6                ; Execbase
        jsr           -$78(a6)         ; Disable
        lea           GfxName(PC),a1         ; Nome lib
        jsr           -$198(a6)         ; OpenLibrary
        move.l         d0,GfxBase
        move.l         d0,a6
        move.l         $26(a6),OldCop        ; salviamo la vecchia COP

;        Puntiamo la PIC "vuota"

        MOVE.L         #BITPLANE,d0         ; dove puntare
        LEA           BPLPOINTERS,A1        ; puntatori COP
        move.w         d0,6(a1)
        swap           d0
        move.w         d0,2(a1)

;        Puntiamo gli sprite 0 ed 1, che ATTACCATI formeranno un solo sprite
;        a 16 colori. Lo sprite1, quello dispari, deve avere il bit 7 della
;        seconda word ad 1.

        MOVE.L         FRAMETAB(PC),d0               ; indirizzo dello sprite in d0
        LEA           SpritePointers,a1        ; Puntatori in copperlist
        move.w         d0,6(a1)
        swap           d0
        move.w         d0,2(a1)
        swap           d0
        ADD.L          #$44,d0               ; lo sprite dispari e' 44 bytes dopo!
        addq.w         #8,a1                        ; prossimi SPRITEPOINTERS
```

```
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

; P.S: non occore settare il bit 7, e' gia' settato nello sprite in questo caso

        move.l        #COPPERLIST,$dff080        ; nostra COP
        move.w        d0,$dff088                ; START COP
        move.w        #0,$dff1fc                ; NO AGA!
        move.w        #$c00,$dff106                ; NO AGA!

mouse:
        cmpi.b        #$ff,$dff006        ; Linea 255?
        bne.s        mouse

        bsr.s        Animazione
        bsr.w        MuoviSprites        ; Muovi gli sprites

Aspetta:
        cmpi.b        #$ff,$dff006        ; linea 255?
        beq.s        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        move.l        OldCop(PC),$dff080        ; Puntiamo la cop di sistema
        move.w        d0,$dff088                ; facciamo partire la vecchia cop

        move.l        4.w,a6
        jsr        -$7e(a6)        ; Enable
        move.l        gfxbase(PC),a1
        jsr        -$19e(a6)        ; Closelibrary
        rts

;        Dati

GfxName:
        dc.b        "graphics.library",0,0

GfxBase:
        dc.l        0

OldCop:
        dc.l        0

; Questa routine anima gli sprite, spostando gli indirizzi dei fotogrammi
; in maniera che ogni volta il primo della tabella vada all'ultimo posto,
; mentra gli altri scorrono tutti di un posto in direzione del primo

Animazione:
        addq.b        #1,ContaAnim    ; queste tre istruzioni fanno si' che il
        cmp.b        #2,ContaAnim    ; fotogramma venga cambiato una volta
        bne.s        NonCambiare     ; si e una no.
        clr.b        ContaAnim
        LEA        FRAMETAB(PC),a0 ; tabella dei fotogrammi
        MOVE.L        (a0),d0                ; salva il primo indirizzo in d0
        MOVE.L        4(a0),(a0)        ; sposta indietro gli altri 5 indirizzi
        MOVE.L        4*2(a0),4(a0)        ; Queste istruzioni "ruotano" gli indirizzi
        MOVE.L        4*3(a0),4*2(a0) ; della tabella.
        MOVE.L        4*4(a0),4*3(a0)
        MOVE.L        4*5(a0),4*4(a0)
        MOVE.L        d0,4*5(a0)        ; metti l'ex primo indirizzo al sesto posto
```

```
        MOVE.L        FRAMETAB(PC),d0               ; indirizzo dello sprite in d0
        LEA         SpritePointers,a1        ; Puntatore sprite pari
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #$44,d0                ; lo sprite dispari e' 44 bytes dopo il pari
        addq.w        #8,a1                 ; POINTER dello sprite dispari
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
NonCambiare:
        rts

ContaAnim:
        dc.w         0

; Questa e‘ la tabella degli indirizzi dei fotogrammi dello sprite pari, da
; cui si accede anche ai rispettivi sprite dispari da attaccare. Gli indirizzi
; presenti nella tabella vengono "ruotati" all'interno della tabella dalla
; routine Animazione, in modo che il primo nella lista sia la prima volta il
; fotogramma1, la volta dopo il Fotogramma2, poi il 3,4,5,6 e di nuovo il
; primo, ciclicamente. In questo modo basta prendere l'indirizzo che sta
; all'inizio della tabella ogni volta dopo il "rimescolamento" per avere gli
; indirizzi dei fotogrammi in sequenza.

FRAMETAB:
        DC.L         Fotogramma1
        DC.L         Fotogramma2
        DC.L         Fotogramma3
        DC.L         Fotogramma4
        DC.L         Fotogramma5
        DC.L         Fotogramma6


; Questa routine legge dalle 2 tabelle le coordinate reali degli sprite.
; poiche‘ gli sprites sono attached hanno entrambi le stesse coordiante.

MuoviSprites:
        ADDQ.L        #1,TABYPOINT          ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte
NOBSTARTY:
        moveq        #0,d3                ; Pulisci d3
        MOVE.b        (A0),d3                ; copia il byte della tabella, cioe‘ la
                          ; coordinata Y in d3

        ADDQ.L        #2,TABXPOINT          ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq        #0,d4                ; azzeriamo d4
        MOVE.w        (A0),d4                ; poniamo il valore della tabella, cioe‘
                          ; la coordinata X in d4
```

```
        MOVE    D3,D0           ; coordinata Y in d0
        MOVE    D4,D1           ; coordinata X in d1
        moveq        #15,d2             ; altezza dello sprite in d2
        MOVE.L        FRAMETAB(PC),a1        ; indirizzo dello sprite in A1

        bsr.w        UniMuoviSprite  ; esegue la routine universale che posiziona
                             ; lo sprite pari

        MOVE.W        D3,D0             ; coordinata Y in d0
        MOVE.W        D4,D1             ; coordinata X in d1
        moveq        #15,d2             ; altezza dello sprite in d2
        LEA        $44(a1),a1        ; indirizzo dello sprite dispari in A1
                             ; lo sprite dispari e‘ $44 bytes dopo quello
                             ; pari
        bsr.w        UniMuoviSprite  ; esegue la routine universale che posiziona
                             ; lo sprite dispari
        rts


TABYPOINT:
        dc.l        TABY-1              ; NOTA: i valori della tabella qua sono bytes,
                             ; dunque lavoriamo con un ADDQ.L #1,TABYPOINT
                             ; e non #2 come per quando sono word o con #4
                             ; come quando sono longword.
TABXPOINT:
        dc.l        TABX-2              ; NOTA: i valori della tabella qua sono word,

; Tabella con coordinate Y dello sprite precalcolate.

TABY:
        incbin        "ycoordinatok.tab"        ; 200 valori .B
FINETABY:

; Tabella con coordinate X dello sprite precalcolate.

TABX:
        incbin        "xcoordinatok.tab"        ; 150 valori .W
FINETABX:



; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W        #$2c,d0              ; aggiungi l’offset dell’inizio dello schermo

; a1 contiene l’indirizzo dello sprite
        MOVE.b        d0,(a1)               ; copia il byte in VSTART
        btst.l        #8,d0
        beq.s        NonVSTARTSET
        bset.b        #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s        ToVSTOP
```

```
NonVSTARTSET:
        bclr.b        #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w        D2,D0                ; Aggiungi l'altezza dello sprite per
                                  ; determinare la posizione finale (VSTOP)
        move.b        d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l        #8,d0
        beq.s        NonVSTOPSET
        bset.b        #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w        VstopFIN
NonVSTOPSET:
        bclr.b        #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w        #128,D1               ; 128 - per centrare lo sprite.
        btst        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                  ; il valore di HSTART, per "trasformarlo" nel
                                  ; valore fa porre nel byte HSTART, senza cioe'
                                  ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts



        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; bit 12 acceso!! 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

;        Palette della PIC

        dc.w        $180,$000        ; color0        ; sfondo nero
        dc.w        $182,$123        ; color1        ; colore 1 del bitplane, che
                                           ; in questo caso e' vuoto,
```

```
                                        ; per cui non compare.

;        Palette degli SPRITE attacched

        dc.w        $1A2,$FFC        ; color17, COL 1 per sprite att.
        dc.w        $1A4,$DEA        ; color18, COL 2 per sprite att.
        dc.w        $1A6,$AC7        ; color19, COL 3 per sprite att.
        dc.w        $1A8,$7B6        ; color20, COL 4 per sprite att.
        dc.w        $1AA,$494        ; color21, COL 5 per sprite att.
        dc.w        $1AC,$284        ; color22, COL 6 per sprite att.
        dc.w        $1AE,$164        ; color23, COL 7 per sprite att.
        dc.w        $1B0,$044        ; color24, COL 7 per sprite att.
        dc.w        $1B2,$023        ; color25, COL 9 per sprite att.
        dc.w        $1B4,$001        ; color26, COL 10 per sprite att.
        dc.w        $1B6,$F80        ; color27, COL 11 per sprite att.
        dc.w        $1B8,$C40        ; color28, COL 12 per sprite att.
        dc.w        $1BA,$820        ; color29, COL 13 per sprite att.
        dc.w        $1BC,$500        ; color30, COL 14 per sprite att.
        dc.w        $1BE,$200        ; color31, COL 15 per sprite att.

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco gli sprite: OVVIAMENTE in CHIP RAM! **********


Fotogramma1:                ; lunghezza 15 linee, $44 bytes
        dc.w $0000,$0000
        dc.w $0580,$0040,$07c0,$0430,$0d68,$0d18,$1fac,$1b9c
        dc.w $3428,$3818,$068e,$993c,$d554,$1390,$729e,$b6d8
        dc.w $5556,$9390,$96b0,$e972,$406c,$7c60,$5bc4,$5fc8
        dc.w $0970,$0908,$0bc0,$0030,$0600,$01c0
        dc.w 0,0
Fotogramma1b:                ; lunghezza 15 linee
        dc.w $0000,$0080
        dc.w $07c0,$0000,$1bf0,$0380,$32f8,$0380,$607c,$0380
        dc.w $43f8,$0384,$e3fc,$0382,$efec,$7ffe,$cfe4,$7ffe
        dc.w $efec,$7ffe,$fff0,$038e,$7fe0,$039c,$5c40,$23bc
        dc.w $0a80,$37f8,$0380,$1ff0,$0000,$07c0
        dc.w 0,0

Fotogramma2:
        dc.w $0000,$0000
        dc.w $0580,$0040,$05c0,$0430,$0ee8,$0e98,$1dac,$1b9c
        dc.w $34e8,$3ad8,$560e,$993c,$f5e8,$3318,$d252,$1690
        dc.w $3a96,$c7d0,$95b8,$ea32,$41ec,$78e0,$5e44,$5e48
        dc.w $0470,$0408,$0ec0,$0030,$0600,$01c0
        dc.w 0,0
Fotogramma2b:
        dc.w $0000,$0080
        dc.w $07c0,$0000,$1bf0,$0180,$3178,$01c0,$607c,$01c0
        dc.w $4138,$01c4,$e3fc,$7382,$cff8,$7f86,$efe0,$fffe
        dc.w $ffec,$0ffe,$ffc8,$07fe,$7ff8,$071c,$5940,$27bc
        dc.w $0a00,$3ff8,$0e00,$1ff0,$0000,$07c0
        dc.w 0,0

Fotogramma3:
        dc.w $0000,$0000
        dc.w $0580,$0040,$04c0,$0430,$0e68,$0e18,$3dfc,$1bec
        dc.w $25c8,$0bd8,$7b2e,$ba3c,$d068,$1798,$6642,$82b0
        dc.w $32d6,$c690,$9490,$eb12,$49bc,$78b0,$4d6c,$4d60
        dc.w $1870,$0808,$0ec0,$0030,$0600,$01c0
```

```
        dc.w 0,0
Fotogramma3b:
        dc.w $0000,$0080
        dc.w $07c0,$0000,$1bf0,$0000,$31f8,$0060,$601c,$20f0
        dc.w $7038,$30e4,$c5dc,$7de2,$eff8,$3fc6,$fff0,$0fce
        dc.w $fff0,$07ee,$ffe8,$07fe,$77c8,$0f7c,$5358,$3ebc
        dc.w $1400,$3ff8,$0c00,$1ff0,$0000,$07c0
        dc.w 0,0

Fotogramma4:
        dc.w $0000,$0000
        dc.w $0580,$0040,$04c0,$0430,$1678,$0608,$357c,$1764
        dc.w $0968,$0968,$122e,$91bc,$c7e8,$0398,$6242,$86b0
        dc.w $3256,$c790,$93b0,$f032,$786c,$5b60,$7354,$4748
        dc.w $1870,$0808,$0ac0,$0030,$0600,$01c0
        dc.w 0,0
Fotogramma4b:
        dc.w $0000,$0080
        dc.w $07c0,$0000,$1bf0,$0000,$39f8,$1830,$689c,$3c78
        dc.w $7698,$3ef4,$efdc,$1fe2,$fff8,$0fc6,$fff0,$07ce
        dc.w $fff0,$0fee,$efc0,$1ffe,$6798,$3cfc,$7f30,$38fc
        dc.w $1820,$37f8,$0000,$1ff0,$0000,$07c0
        dc.w 0,0

Fotogramma5:
        dc.w $0000,$0000
        dc.w $0580,$0040,$04c0,$0030,$0e68,$0218,$172c,$1714
        dc.w $3ca8,$3ca0,$0116,$9810,$cf10,$09d0,$64e2,$8290
        dc.w $30d6,$d7b0,$8a50,$c992,$782c,$5b20,$7be4,$4fe8
        dc.w $0830,$0808,$0ae0,$0010,$0600,$01c0
        dc.w 0,0
Fotogramma5b:
        dc.w $0000,$0080
        dc.w $07c0,$0000,$1ff0,$0400,$3df8,$0c00,$68fc,$0e08
        dc.w $4358,$0f3c,$e7ec,$077e,$f7e8,$07fe,$fff0,$07ee
        dc.w $eff0,$1fce,$f7f0,$3fee,$67c0,$3cfc,$7f10,$30fc
        dc.w $0870,$37f8,$0060,$1ff0,$0000,$07c0
        dc.w 0,0

Fotogramma6:
        dc.w $0000,$0000
        dc.w $0580,$0040,$07c0,$0430,$0e68,$0a18,$1b2c,$1b1c
        dc.w $3428,$3c18,$0696,$9910,$cf5c,$0d98,$7492,$92d0
        dc.w $50b6,$97d0,$ab70,$c8b2,$602c,$5e20,$5bc4,$5fc8
        dc.w $0850,$0848,$0ae0,$0010,$0600,$01c0
        dc.w 0,0
Fotogramma6b:
        dc.w $0000,$0080
        dc.w $07c0,$0000,$1bf0,$0300,$35f8,$0700,$64fc,$0700
        dc.w $43f8,$0784,$e3ec,$03be,$f3e4,$03fe,$efee,$1ffe
        dc.w $eff0,$7fee,$f7f0,$3fce,$7fe0,$21dc,$5e00,$21fc
        dc.w $08a0,$37f8,$00e0,$1ff0,$0000,$07c0
        dc.w 0,0


        SECTION        PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                               ; perche' per vedere gli sprite
                                               ; e' necessario che ci siano bitplanes
                                               ; abilitati
BITPLANE:
        ds.b        40*256                ; bitplane azzerato lowres
```

```
        end
```

In questo esempio mostriamo come realizzare uno sprite animato, seguendo la
tecnica spiegata nella lezione.
La figura che animiamo e‘ formata da una coppia di sprite "attached".
In pratica quindi sono 2 gli sprite che animiamo.
Per ciascun sprite abbiamo 6 fotogrammi. Consideriamo per ora solo il primo
sprite. Ogni fotogramma e‘ memorizzato in una struttura sprite.
Ogni volta che lo sprite viene ridisegnato, la routine "animazione" provvede
ad utilizzare un diverso fotogramma, ovvero una diversa struttura sprite.
La routine infatti gestisce una tabella con gli indirizzi delle varie
strutture sprite, e ogni volta che viene eseguita sposta gli indirizzi
all’interno della tabella in maniera che tutti, a rotazione vengano a trovarsi
all’inizio della tabella.
In pratica non c’e’ nulla di nuovo, perche’ siamo di fronte ad una tabella
di indirizzi, anziche’ una tabella di valori. Inoltre i 6 indirizzi vengono
"ruotati" all’interno della tabella stessa, cioe’ ogni fotogramma il primo
indirizzo viene messo al posto dell’ultimo, il secondo al posto del primo, il
terzo al posto del secondo e cosi’ via, in modo analogo alla rotazione che
abbiamo gia’ visto per i colori in copperlist nella Lezione3e.s.
L’indirizzo che si trova in testa alla tabella viene caricato nel puntatore
dello sprite e usato come fotogramma per lo sprite.
Per evitare di ripetere questo lavoro anche per il secondo sprite (quello
dispari da attaccare al primo), ogni fotogramma di ogni "secondo sprite" e‘
sistemato in memoria subito dopo il corrispondente fotogramma del primo sprite,
in modo tale che dall’indirizzo del fotogramma del primo sprite (pari) si puo‘
risalire all’indirizzo del corrispondente secondo sprite (dispari) da attaccare
al primo semplicemente con un:

```
        lea     $44(a0),a1
```

Che aggiunge all’indirizzo del fotogramma del primo sprite la lunghezza del
fotogramma stesso, ottenendo l’indirizzo del secondo fotogramma (dispari).

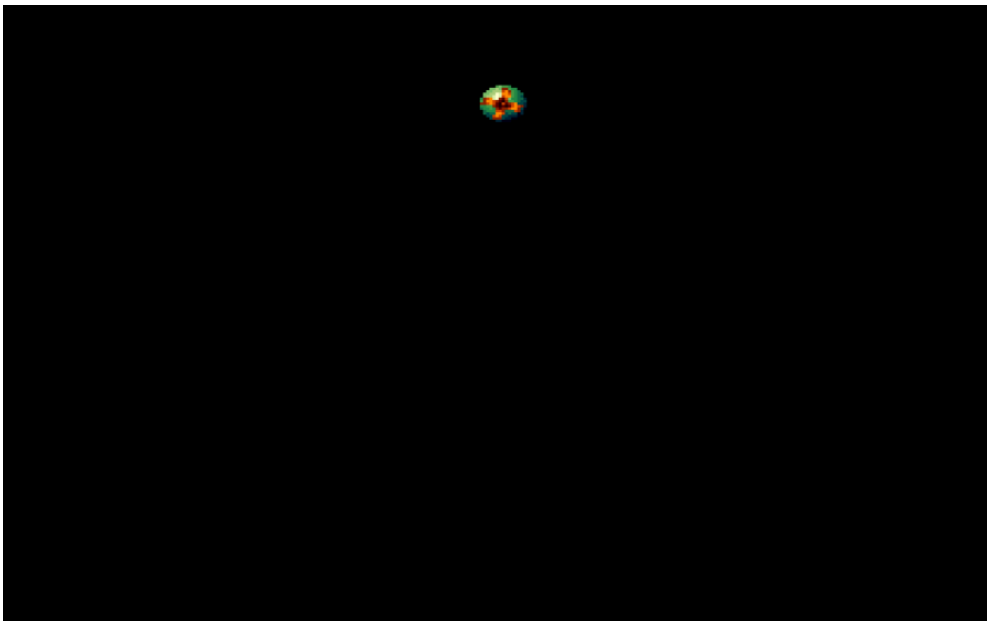


Figura 23.12: Lezione 7z

CAPITOLO 24

LEZIONE 8

24.1 Lezione8a

```
; Lezione8a.s - La startup universale, per studiare i canali DMA

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                  -----a-bcdefghij

;        a: Blitter Nasty   (Per ora non ci interessa, lasciamolo a zero)
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA           (Azzerandolo non e' eseguita nemmeno la copperlist)
;        d: Blitter DMA           (Per ora non ci interessa, azzeriamolo)
;        e: Sprite DMA           (Azzerandolo spariscono solo gli 8 sprite)
;        f: Disk DMA           (Per ora non ci interessa, azzeriamolo)
;        g-j: Audio 3-0 DMA (Azzeriamo lasciando muto l'Amiga)

*****************************************************************************
;    680X0 & AGA STARTUP BY FABIO CIUCCI - Livello di complessita' 1
*****************************************************************************

MAINCODE:
        movem.l        d0-d7/a0-a6,-(SP)        ; Salva i registri nello stack
        move.l        4.w,a6                        ; ExecBase in a6
        LEA        GfxName(PC),A1                ; Nome libreria da aprire
        JSR        -$198(A6)                ; OldOpenLibrary - apri la lib
        MOVE.L        d0,GFXBASE                ; Salva il GfxBase in una label
        BEQ.w        EXIT2                        ; Se si, esci senza eseguire il codice
        LEA        IntuiName(PC),A1        ; Intuition.lib
        JSR        -$198(A6)                ; Openlib
        MOVE.L        D0,IntuiBase
        BEQ.w        EXIT1                        ; Se zero, esci! Errore!

        MOVE.L        IntuiBase(PC),A0
```

```
        CMP.W       #39,$14(A0)        ; versione 39 o maggiore? (kick3.0+)
        BLT.s       VecchiaIntui
        BSR.w       ResettaSpritesV39
VecchiaIntui:

        MOVE.L       GfxBase(PC),A6
        MOVE.L       $22(A6),WBVIEW        ; Salva il WBView attuale di sistema

        SUBA.L       A1,A1                ; View nullo per azzerare il modo video
        JSR        -$DE(A6)        ; LoadView nullo - modo video azzerato
        SUBA.L       A1,A1                ; View nullo
        JSR        -$DE(A6)        ; LoadView (due volte per sicurezza...)
        JSR        -$10E(A6)        ; WaitOf ( Queste due chiamate a WaitOf    )
        JSR        -$10E(A6)        ; WaitOf ( servono a resettare l'interlace )
        JSR        -$10E(A6)        ; Altre due, vah!
        JSR        -$10E(A6)

        MOVEA.L       4.w,A6
        SUBA.L       A1,A1                ; NULL task - trova questo task
        JSR        -$126(A6)        ; findtask (d0=task, FindTask(name) in a1)
        MOVEA.L       D0,A1                ; Task in a1
        MOVEQ       #127,D0                ; Priorita' in d0 (-128, +127) - MASSIMA!
        JSR        -$12C(A6)        ;_LVOSetTaskPri (d0=priorita', a1=task)

        MOVE.L       GfxBase(PC),A6
        jsr        -$1c8(a6)        ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.
        jsr        -$E4(A6)        ; WaitBlit - Attende la fine di ogni blittata
        JSR        -$E4(A6)        ; WaitBlit

        move.l       4.w,a6                ; ExecBase in A6
        JSR        -$84(a6)        ; FORBID - Disabilita il Multitasking
        JSR        -$78(A6)        ; DISABLE - Disabilita anche gli interrupt
                                ;            del sistema operativo

        bsr.w       HEAVYINIT        ; Ora puoi eseguire la parte che opera
                                ; sui registri hardware

        move.l       4.w,a6                ; ExecBase in A6
        JSR        -$7E(A6)        ; ENABLE - Abilita System Interrupts
        JSR        -$8A(A6)        ; PERMIT - Abilita il multitasking

        SUBA.L       A1,A1                ; NULL task - trova questo task
        JSR        -$126(A6)        ; findtask (d0=task, FindTask(name) in a1)
        MOVEA.L       D0,A1                ; Task in a1
        MOVEQ       #0,D0                ; Priorita' in d0 (-128, +127) - NORMALE
        JSR        -$12C(A6)        ;_LVOSetTaskPri (d0=priorita', a1=task)

        MOVE.W       #$8040,$DFF096        ; abilita blit
        BTST.b       #6,$dff002        ; WaitBlit...
Wblittez:
        BTST.b       #6,$dff002
        BNE.S       Wblittez

        MOVE.L       GFXBASE(PC),A6        ; GFXBASE in A6
        jsr        -$E4(A6)        ; Aspetta la fine di eventuali blittate
        JSR        -$E4(A6)        ; WaitBlit
        jsr        -$1ce(a6)        ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
        MOVE.L       IntuiBase(PC),A0
        CMP.W       #39,$14(A0)        ; V39+?
        BLT.s       Vecchissima
```

```
        BSR.w        RimettiSprites
Vecchissima:

        MOVE.L        GFXBASE(PC),A6        ; GFXBASE in A6
        MOVE.L        $26(a6),$dff080        ; COP1LC - Punta la vecchia copper1 di sistema
        MOVE.L        $32(a6),$dff084        ; COP2LC - Punta la vecchia copper2 di sistema
        JSR        -$10E(A6)        ; WaitOf ( Risistema l'eventuale interlace)
        JSR        -$10E(A6)        ; WaitOf
        MOVE.L        WBVIEW(PC),A1        ; Vecchio WBVIEW in A1
        JSR        -$DE(A6)        ; loadview - rimetti il vecchio View
        JSR        -$10E(A6)        ; WaitOf ( Risistema l'eventuale interlace)
        JSR        -$10E(A6)        ; WaitOf
        MOVE.W        #$11,$DFF10C        ; Questo non lo ripristina da solo..!
        MOVE.L        $26(a6),$dff080        ; COP1LC - Punta la vecchia copper1 di sistema
        MOVE.L        $32(a6),$dff084        ; COP2LC - Punta la vecchia copper2 di sistema
        moveq        #100,d7
RipuntLoop:
        MOVE.L        $26(a6),$dff080        ; COP1LC - Punta la vecchia copper1 di sistema
        move.w        d0,$dff088
        dbra        d7,RipuntLoop        ; Per sicurezza...

        MOVEA.L        IntuiBase(PC),A6
        JSR        -$186(A6)        ; _LVORethinkDisplay - Ridisegna tutto il
                                ; display, comprese ViewPorts e eventuali
                                ; modi interlace o multisync.
        MOVE.L        a6,A1                ; IntuiBase in a1 per chiudere la libreria
        move.l        4.w,a6                ; ExecBase in A6
        jsr        -$19E(a6)        ; CloseLibrary - intuition.library CHIUSA
Exit1:
        MOVE.L        GfxBase(PC),A1        ; GfxBase in a1 per chiudere la libreria
        jsr        -$19E(a6)        ; CloseLibrary - graphics.library CHIUSA
Exit2:
        movem.l        (SP)+,d0-d7/a0-a6 ; Riprendi i vecchi valori dei registri
        RTS                        ; Torna all'ASMONE o al Dos/WorkBench

*****************************************************************************
;        Resetta la risoluzione degli sprite "legalmente"
*****************************************************************************

ResettaSpritesV39:
        LEA        Workbench(PC),A0 ; Nome schermo del Workbench in a0
        MOVE.L        IntuiBase(PC),A6
        JSR        -$1FE(A6)        ; _LVOLockPubScreen - "blocchiamo" lo schermo
                                ; (il cui nome e' in a0).
        MOVE.L        D0,SchermoWBLocckato
        BEQ.s        ErroreSchermo
        MOVE.L        D0,A0                ; Struttura Screen in a0
        MOVE.L        $30(A0),A0        ; sc_ViewPort+vp_ColorMap: in a0 ora abbiamo
                                ; la struttura ColorMap dell schermo, che ci
                                ; serve (in a0) per eseguire un "video_control"
                                ; della graphics.library.
        LEA        GETVidCtrlTags(PC),A1        ; In a1 la TagList per la routine
                                        ; "Video_control" - la richiesta che
                                        ; facciamo a questa routine e' di
                                        ; VTAG_SPRITERESN_GET, ossia di sapere
                                        ; la risoluzione attuale degli sprite.
        MOVE.L        GfxBase(PC),A6
        JSR        -$2C4(A6)        ; Video_Control (in a0 la cm e in a1 i tags)
                                ; riporta nella taglist, nella long
                                ; "risoluzione", la risoluzione attuale degli
                                ; sprite in quello schermo.
```

```
; Ora chiediamo alla routine VideoControl di settare la risoluzione.
; SPRITERESN_140NS -> ossia lowres!

        MOVE.L        SchermoWBLocckato(PC),A0
        MOVE.L        $30(A0),A0        ; struttura sc_ViewPort+vp_ColorMap in a0
        LEA        SETVidCtrlTags(PC),A1        ; TagList che resetta gli sprite.
        MOVE.L        GfxBase(PC),A6
        JSR        -$2C4(A6)        ; video_control... resetta gli sprite!

; Ora resettiamo anche l'eventuale schermo "in primo piano", ad esempio la
; schermata dell'assemblatore:

        MOVE.L        IntuiBase(PC),A6
        move.l        $3c(a6),a0        ; Ib_FirstScreen (Schermo in "primo piano!")
        MOVE.L        $30(A0),A0        ; struttura sc_ViewPort+vp_ColorMap in a0
        LEA        GETVidCtrlTags2(PC),A1        ; In a1 la TagList GET
        MOVE.L        GfxBase(PC),A6
        JSR        -$2C4(A6)        ; Video_Control (in a0 la cm e in a1 i tags)

        MOVEA.L        IntuiBase(PC),A6
        move.l        $3c(a6),a0        ; Ib_FirstScreen - "pesca" lo schermo in
                                ; primo piano (ad es. ASMONE)
        MOVEA.L        $30(A0),A0        ; struttura sc_ViewPort+vp_ColorMap in a0
        LEA        SETVidCtrlTags(PC),A1        ; TagList che resetta gli sprite.
        MOVEA.L        GfxBase(PC),A6
        JSR        -$2C4(A6)        ; video_control... resetta gli sprite!

        MOVEA.L        SchermoWBLocckato(PC),A0
        MOVEA.L        IntuiBase(PC),A6
        JSR        -$17A(A6)        ; _LVOMakeScreen - occorre rifare lo schermo
        move.l        $3c(a6),a0        ; Ib_FirstScreen - "pesca" lo schermo in
                                ; primo piano (ad es. ASMONE)
        JSR        -$17A(A6)        ; _LVOMakeScreen - occorre rifare lo schermo
                                ; per essere sicuri del reset: ossia occorre
                                ; chiamare MakeScreen, seguito da...
        JSR        -$186(A6)        ; _LVORethinkDisplay - che ridisegna tutto il
                                ; display, comprese ViewPorts e eventuali
ErroreSchermo:                        ; modi interlace o multisync.
        RTS

; Ora dobbiamo risettare gli sprites alla risoluzione di partenza.

RimettiSprites:
        MOVE.L        SchermoWBLocckato(PC),D0 ; Indirizzo strutt. Screen
        BEQ.S        NonAvevaFunzionato        ; Se = 0, allora peccato...
        MOVE.L        D0,A0
        MOVE.L        OldRisoluzione(PC),OldRisoluzione2 ; Rimetti vecchia risoluz.
        LEA        SETOldVidCtrlTags(PC),A1
        MOVE.L        $30(A0),A0        ; Struttura ColorMap dello screen
        MOVE.L        GfxBase(PC),A6
        JSR        -$2C4(A6)        ; _LVOVideoControl - Risetta la risoluzione

; Ora dello schermo in primo piano (eventuale)...

        MOVE.L        IntuiBase(PC),A6
        move.l        $3c(a6),a0        ; Ib_FirstScreen - "pesca" lo schermo in
                                ; primo piano (ad es. ASMONE)
        MOVE.L        OldRisoluzioneP(PC),OldRisoluzione2 ; Rimetti vecchia risoluz.
        LEA        SETOldVidCtrlTags(PC),A1
        MOVE.L        $30(A0),A0        ; Struttura ColorMap dello screen
        MOVE.L        GfxBase(PC),A6
        JSR        -$2C4(A6)        ; _LVOVideoControl - Risetta la risoluzione
```

```
        MOVEA.L        SchermoWBLocckato(PC),A0
        MOVEA.L        IntuiBase(PC),A6
        JSR        -$17A(A6)        ; RethinkDisplay - "ripensiamo" il display
        move.l        $3c(a6),a0        ; Ib_FirstScreen - schermo in primo piano
        JSR        -$17A(A6)        ; RethinkDisplay - "ripensiamo" il display
        MOVE.L        SchermoWBLocckato(PC),A1
        SUB.L        A0,A0                ; null
        MOVEA.L        IntuiBase(PC),A6
        JSR        -$204(A6)        ; _LVOUnlockPubScreen - e "sblocchiamo" lo
NonAvevaFunzionato:                ; screen del workbench.
        RTS

SchermoWBLocckato:
        dc.l        0

; Questa e' la struttura per usare Video_Control. La prima long serve per
; CAMBIARE (SET) la risoluzione degli sprite o per sapere (GET) quella vecchia.

GETVidCtrlTags:
        dc.l        $80000032        ; GET
OldRisoluzione:
        dc.l        0        ; Risoluzione sprite: 0=ECS, 1=lowres, 2=hires, 3=shres
        dc.l        0,0,0        ; 3 zeri per TAG_DONE (terminare la TagList)

GETVidCtrlTags2:
        dc.l        $80000032        ; GET
OldRisoluzioneP:
        dc.l        0        ; Risoluzione sprite: 0=ECS, 1=lowres, 2=hires, 3=shres
        dc.l        0,0,0        ; 3 zeri per TAG_DONE (terminare la TagList)

SETVidCtrlTags:
        dc.l        $80000031        ; SET
        dc.l        1        ; Risoluzione sprite: 0=ECS, 1=lowres, 2=hires, 3=shres
        dc.l        0,0,0        ; 3 zeri per TAG_DONE (terminare la TagList)

SETOldVidCtrlTags:
        dc.l        $80000031        ; SET
OldRisoluzione2:
        dc.l        0        ; Risoluzione sprite: 0=ECS, 1=lowres, 2=hires, 3=shres
        dc.l        0,0,0        ; 3 zeri per TAG_DONE (terminare la TagList)

; Nome schermo del WorkBench

Workbench:
        dc.b        'Workbench',0

*****************************************************************************
;        Da qua in avanti si puo' operare sull'hardware in modo diretto
*****************************************************************************

HEAVYINIT:
        LEA        $DFF000,A5                ; Base dei registri CUSTOM per Offsets
        MOVE.W        $2(A5),OLDDMA                ; Salva il vecchio status di DMACON
        MOVE.W        $1C(A5),OLDINTENA        ; Salva il vecchio status di INTENA
        MOVE.W        $10(A5),OLDADKCON        ; Salva il vecchio status di ADKCON
        MOVE.W        $1E(A5),OLDINTREQ        ; Salva il vecchio status di INTREQ
        MOVE.L        #$80008000,d0                ; Prepara la maschera dei bit alti
                                        ; da settare nelle word dove sono
                                        ; stati salvati i registri
        OR.L        d0,OLDDMA        ; Setta il bit 15 di tutti i valori salvati
        OR.L        d0,OLDADKCON        ; dei registri hardware, indispensabile per
```

```
                            ; rimettere tali valori nei registri.

        MOVE.L        #$7FFF7FFF,$9A(a5)        ; DISABILITA GLI INTERRUPTS & INTREQS
        MOVE.L        #0,$144(A5)                ; SPRODAT - ammazza il puntatore!
        MOVE.W        #$7FFF,$96(a5)                ; DISABILITA I DMA

        bsr.s        START                        ; Esegui il programma.

        LEA        $dff000,a5                ; Custom base per offsets
        MOVE.W        #$7FFF,$96(A5)                ; DISABILITA TUTTI I DMA
        MOVE.L        #$7FFF7FFF,$9A(A5)        ; DISABILITA GLI INTERRUPTS & INTREQS
        MOVE.W        #$7fff,$9E(a5)                ; Disabilita i bit di ADKCON
        MOVE.W        OLDADKCON(PC),$9E(A5)        ; ADKCON
        MOVE.W        OLDDMA(PC),$96(A5)        ; Rimetti il vecchio status DMA
        MOVE.W        OLDINTENA(PC),$9A(A5)        ; INTENA STATUS
        MOVE.W        OLDINTREQ(PC),$9C(A5)        ; INTREQ
        RTS

;        Dati salvati dalla startup

WBVIEW:                        ; Indirizzo del View del WorkBench
        DC.L        0
GfxName:
        dc.b        'graphics.library',0,0
IntuiName:
        dc.b        'intuition.library',0

GfxBase:                ; Puntatore alla Base della Graphics Library
        dc.l        0
IntuiBase:                ; Puntatore alla Base della Intuition Library
        dc.l        0
OLDDMA:                        ; Vecchio status DMACON
        dc.w        0
OLDINTENA:                ; Vecchio status INTENA
        dc.w        0
OLDADKCON:                ; Vecchio status ADKCON
        DC.W        0
OLDINTREQ:                ; Vecchio status INTREQ
        DC.W        0

START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane e copper

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

mouse:
        btst        #6,$bfe001
        bne.s        mouse
        rts
```

```
        Section        CopProva,data_C

COPPERLIST:
        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE

;        Sfumatura copperlist

        dc.w        $5007,$fffe        ; WAIT linea $50
        dc.w        $180,$001        ; color0
        dc.w        $5207,$fffe        ; WAIT linea $52
        dc.w        $180,$002        ; color0
        dc.w        $5407,$fffe        ; WAIT linea $54
        dc.w        $180,$003        ; color0
        dc.w        $5607,$fffe        ; WAIT linea $56
        dc.w        $180,$004        ; color0
        dc.w        $5807,$fffe        ; WAIT linea $58
        dc.w        $180,$005        ; color0
        dc.w        $5a07,$fffe        ; WAIT linea $5a
        dc.w        $180,$006        ; color0
        dc.w        $5c07,$fffe        ; WAIT linea $5c
        dc.w        $180,$007        ; color0
        dc.w        $5e07,$fffe        ; WAIT linea $5e
        dc.w        $180,$008        ; color0
        dc.w        $6007,$fffe        ; WAIT linea $60
        dc.w        $180,$009        ; color0
        dc.w        $6207,$fffe        ; WAIT linea $62
        dc.w        $180,$00a        ; color0

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;        Con il comando dcb facciamo un disegno a caso per il bitplane

BITPLANE:
        dcb.l        10240/4,$FF00FF00

        end
```

Ci sono due particolari che non figurano nella lezione: il primo e' che c'e'
una routine che resetta il modo video degli sprite, in caso il kickstart sia
il 3.0 o superiore.
Il secondo particolare e' che e' stata aggiunta una istruzione a quelle
presenti per disabilitare l'AGA:

```
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA
```

In realta' anche questa istruzione e' quasi superflua, perche' quasi mai e'

```
fuori posto, ma anche qua la sicurezza non deve essere trascurata.

Provate a spegnere i canali dma dei bitplane e del copper, uno ad uno.
Noterete che spegnendo solo il canale dei bitplane sparisce il disegno
a barre, disabilitando il copper sparisce pure la sfumatura.
Provate anche a disattivare solo il bit 9, l'interruttore generale, e vedrete
che anche se gli atri bit sono attivati si spegne tutto.
Inutile provare ad azzerare il bit 15!
```

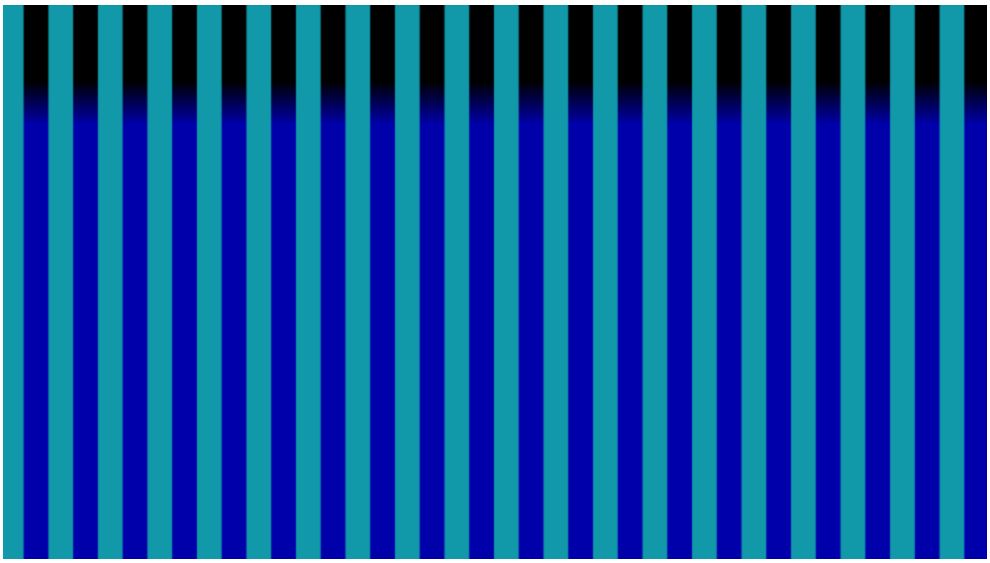


Figura 24.1: Lezione 8a

24.2 Lezione8b

```
; Lezione8b.s - Utilizzo della startup universale per un esempio che e'
;                una fusione di Lezione7o.s degli sprite e della routine di
;                print della lezione 6.


        Section        UsoLaStartUp,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110100000        ; copper e bitplane DMA abilitati
;                  -----a-bcdefghij
```

```
;        a: Blitter Nasty   (Per ora non ci interessa, lasciamolo a zero)
;        b: Bitplane DMA          (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA          (Azzerandolo non e' eseguita nemmeno la copperlist)
;        d: Blitter DMA          (Per ora non ci interessa, azzeriamolo)
;        e: Sprite DMA          (Azzerandolo spariscono solo gli 8 sprite)
;        f: Disk DMA          (Per ora non ci interessa, azzeriamolo)
;        g-j: Audio 3-0 DMA (Azzeriamo lasciando muto l'Amiga)

; MAIN PROGRAM - ricordarsi che i canali DMA sono tutti azzerati

START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

;        Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L        #SpriteNullo,d0               ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        MOVEQ        #8-1,d1                       ; tutti gli 8 sprite
NulLoop:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        addq.w        #8,a1
        dbra        d1,NulLoop

;        Puntiamo lo sprite

        MOVE.L        #MIOSPRITE,d0               ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        addq.w        #8,a1                       ; puntatore a sprite 1
        MOVE.L        #MIOSPRITE2,d0               ; indirizzo dello sprite in d0
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)               ; DMACON - abilita bitplane, copper
                                    ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
```

```
        CMPI.L       D2,D0               ; aspetta la linea $130 (304)
        BNE.S       Waity1

        bsr.s       PrintCarattere        ; Stampa un carattere alla volta
        bsr.w       MuoviSprite        ; Muovi gli sprite 0 ed 1

        MOVE.L       #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L       #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L       4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L       D1,D0               ; Seleziona solo i bit della pos. verticale
        CMPI.L       D2,D0               ; aspetta la linea $130 (304)
        BEQ.S       Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s       mouse
        rts                       ; esci

*****************************************************************************
;                       Routine di Print
*****************************************************************************

PRINTcarattere:
        MOVE.L       PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ       #0,D2                ; Pulisci d2
        MOVE.B       (A0)+,D2        ; Prossimo carattere in d2
        CMP.B       #$ff,d2               ; Segnale di fine testo? ($FF)
        beq.s       FineTesto        ; Se si, esci senza stampare
        TST.B       d2                ; Segnale di fine riga? ($00)
        bne.s       NonFineRiga        ; Se no, non andare a capo

        ADD.L       #40*7,PuntaBITPLANE        ; ANDIAMO A CAPO
        ADDQ.L       #1,PuntaTesto                ; primo carattere riga dopo
                                  ; (saltiamo lo ZERO)
        move.b       (a0)+,d2               ; primo carattere della riga dopo
                                  ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B       #$20,D2               ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                          ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                          ; DELLO SPAZIO (che e' $20), in $00, quello
                          ; DELL'ASTERISCO ($21), in $01...
        LSL.W       #3,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                          ; essendo i caratteri alti 8 pixel
        MOVE.L       D2,A2
        ADD.L       #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L       PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                          ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B       (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B       (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B       (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B       (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B       (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B       (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B       (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B       (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.L       #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L       #1,PuntaTesto        ; prossimo carattere da stampare
```

```
FineTesto:
        RTS


PuntaTesto:
        dc.l        TESTO

PuntaBitplane:
        dc.l        BITPLANE

;        $00 per "fine linea" - $FF per "fine testo"

               ; numero caratteri per linea: 40
TESTO:            ;                 11111111112222222223333333334
            ;   12345678901234567890123456789012345678901234567890
        dc.b        '                                        ',0 ; 1
        dc.b        '    Questo listato utilizza i canali    ',0 ; 2
        dc.b        '                                        ',0 ; 3
        dc.b        '    DMA del COPPER, dei BITPLANE e      ',0 ; 4
        dc.b        '                                        ',0 ; 5
        dc.b        '    degli SPRITE, provate a non         ',0 ; 6
        dc.b        '                                        ',0 ; 7
        dc.b        '    abilitarli uno ad uno e vedrete     ',0 ; 8
        dc.b        '                                        ',0 ; 9
        dc.b        '    sparire gli sprite, poi il testo    ',0 ; 10
        dc.b        '                                        ',0 ; 11
        dc.b        '    e anche le sfumature del copper!    ',$FF ; 12

        EVEN

*****************************************************************************
;        Routines degli sprite
*****************************************************************************

MuoviSprite:
        ADDQ.L        #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte
NOBSTARTY:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte della tabella, cioe' la
                            ; coordinata Y in d0 in modo da farla
                            ; trovare alla routine universale

        ADDQ.L        #2,TABXPOINT         ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                            ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq        #0,d1                ; azzeriamo d1
        MOVE.w        (A0),d1                ; poniamo il valore della tabella, cioe'
                            ; la coordinata X in d1

        lea        MIOSPRITE,a1        ; indirizzo dello sprite in A1
        moveq        #13,d2                ; altezza dello sprite in d2

        bsr.w        UniMuoviSprite  ; esegue la routine universale che posiziona
```

```
                              ; lo sprite
; secondo sprite

        ADDQ.L        #1,TABYPOINT2         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT2(PC),A0 ; indirizzo contenuto in long TABXPOINT
                               ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S        NOBSTARTY2        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT2 ; Riparti a puntare dal primo byte
NOBSTARTY2:
        moveq        #0,d0                ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte della tabella, cioe' la
                              ; coordinata Y in d0 in modo da farla
                              ; trovare alla routine universale

        ADDQ.L        #2,TABXPOINT2         ; Fai puntare alla word successiva
        MOVE.L        TABXPOINT2(PC),A0 ; indirizzo contenuto in long TABXPOINT
                               ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX2        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT2 ; Riparti a puntare dalla prima word-2
NOBSTARTX2:
        moveq        #0,d1                ; azzeriamo d1
        MOVE.w        (A0),d1                ; poniamo il valore della tabella, cioe'
                              ; la coordinata X in d1

        lea        MIOSPRITE2,a1        ; indirizzo dello sprite in A1
        moveq        #8,d2                ; altezza dello sprite in d2

        bsr.w        UniMuoviSprite  ; esegue la routine universale che posiziona
                              ; lo sprite
        rts

; puntatori alle tabelle del primo sprite

TABYPOINT:
        dc.l        TABY-1
TABXPOINT:
        dc.l        TABX-2

; puntatori alle tabelle del secondo sprite

TABYPOINT2:
        dc.l        TABY+40-1
TABXPOINT2:
        dc.l        TABX+96-2

; Tabella con coordinate Y dello sprite precalcolate.
TABY:
        incbin        "ycoordinatok.tab"        ; 200 valori .B
FINETABY:

; Tabella con coordinate X dello sprite precalcolate.
TABX:
        incbin        "xcoordinatok.tab"        ; 150 valori .W
FINETABX:

; Routine universale di posizionamento degli sprite.

;
;        Parametri in entrata di UniMuoviSprite:
;
```

```
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W        #$2c,d0                ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b        d0,(a1)                ; copia il byte in VSTART
        btst.l        #8,d0
        beq.s        NonVSTARTSET
        bset.b        #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s        ToVSTOP
NonVSTARTSET:
        bclr.b        #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w        D2,D0                ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b        d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l        #8,d0
        beq.s        NonVSTOPSET
        bset.b        #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w        VstopFIN
NonVSTOPSET:
        bclr.b        #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w        #128,D1                ; 128 - per centrare lo sprite.
        btst        #0,D1                ; bit basso della coordinata X azzerato?
        beq.s        BitBassoZERO
        bset        #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s        PlaceCoords

BitBassoZERO:
        bclr        #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w        #1,D1                ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b        D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

*****************************************************************************

;        Il FONT caratteri 8x8 copiato in CHIP dalla CPU e non dal blitter,
;        per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin        "assembler2:sorgenti4/nice.fnt"

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
```

```
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$19a        ; color1 - SCRITTE

        dc.w        $1A2,$F00        ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w        $1A4,$0F0        ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w        $1A6,$FF0        ; color19, ossia COLOR3 dello sprite0 - GIALLO

;        Sfumatura copperlist

        dc.w        $5007,$fffe        ; WAIT linea $50
        dc.w        $180,$001        ; color0
        dc.w        $5207,$fffe        ; WAIT linea $52
        dc.w        $180,$002        ; color0
        dc.w        $5407,$fffe        ; WAIT linea $54
        dc.w        $180,$003        ; color0
        dc.w        $5607,$fffe        ; WAIT linea $56
        dc.w        $180,$004        ; color0
        dc.w        $5807,$fffe        ; WAIT linea $58
        dc.w        $180,$005        ; color0
        dc.w        $5a07,$fffe        ; WAIT linea $5a
        dc.w        $180,$006        ; color0
        dc.w        $5c07,$fffe        ; WAIT linea $5c
        dc.w        $180,$007        ; color0
        dc.w        $5e07,$fffe        ; WAIT linea $5e
        dc.w        $180,$008        ; color0
        dc.w        $6007,$fffe        ; WAIT linea $60
        dc.w        $180,$009        ; color0
        dc.w        $6207,$fffe        ; WAIT linea $62
        dc.w        $180,$00a        ; color0


        dc.w        $FFFF,$FFFE        ; Fine della copperlist


; ************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! ********

SpriteNullo:                        ; Sprite nullo da puntare in copperlist
        dc.l        0,0,0,0               ; negli eventuali puntatori inutilizzati


MIOSPRITE:                ; lunghezza 13 linee
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.b $5d        ; $50+13=$5d        ; posizione verticale di fine sprite
```

```
        dc.b $00
 dc.w        %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w        %0000000000000000,%0000011001100000
 dc.w        %0000000000000000,%0000001001000000
 dc.w        %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w        %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w        %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w        %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w        %0000011111100000,%0000011111100000
 dc.w        %0000011111100000,%0001111001111000
 dc.w        %0000001111000000,%0011101111011100
 dc.w        %0000000110000000,%0011000110001100
 dc.w        %0000000000000000,%1111000000001111
 dc.w        %0000000000000000,%1111000000001111
 dc.w        0,0         ; 2 word azzerate definiscono la fine dello sprite.


MIOSPRITE2:                 ; lunghezza 8 linee
VSTART2:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART2:
        dc.b $60+(14*2)         ; Pos. orizzontale (da $40 a $d8)
VSTOP2:
        dc.b $68        ; $60+8=$68         ; fine verticale.
        dc.b $00
 dc.w        %0000001111000000,%0111110000111110
 dc.w        %0000011111110000,%1111000111001111
 dc.w        %0011111111111100,%1100001000100011
 dc.w        %0111111111111110,%1000000000100001
 dc.w        %0111111111111110,%1000000111000001
 dc.w        %0011111111111100,%1100001000000011
 dc.w        %0000011111110000,%1111001111101111
 dc.w        %0000001111000000,%0111110000111110
 dc.w        0,0         ; fine sprite


*****************************************************************************

        SECTION         MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256          ; un bitplane lowres 320x256

        end

In questo listato compaiono 2 ottimizzazioni di routines gia' viste.
Una e' quella di cui si parla nella Lezione8, ossia l'attesa della linea
verticale:

        MOVE.L          #$1ff00,d1          ; bit per la selezione tramite AND
        MOVE.L          #$13000,d2          ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L          4(A5),D0          ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L          D1,D0                   ; Seleziona solo i bit della pos. verticale
        CMPI.L          D2,D0                   ; aspetta la linea $130 (304)
        BNE.S          Waity1

Vi ricordo che la linea massima che potete attendere e' $138, se attendete
la $139 o oltre la routine si blocca perche' non si verifica mai tale valore.

L'altra ottimizzazione, che forse e' passata inosservata, e' un:
```

```
        MULU.W        #8,d2
```

Che e' stato trasformato in:

```
        LSL.W         #3,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                  ; essendo i caratteri alti 8 pixel
```

Nella routine di PRINT. Ebbene, spostare a sinistra di 3 bit corrisponde a moltiplocare per 8, come spostare a sinistra di 1 bit significa moltiplicare per 2 e spostare a sinistra di 2 bit significa moltiplicare per 4.
Questo e' perche' il binario agevola le moltiplicazioni e le divisioni per potenze di 2. Vediamo un esempio:

```
        5 * 8 = 40
```

Vediamo in binario:

```
        %00000101 * %00001000 = %00101000
```

Come vedete il risultato, 40, e' lo stesso del 5, ma con i bit spostati a sinistra di 3 posizioni. Vedremo in seguito molti di questi stratagemmi per velocizzare il codice; la cosa piu' importante da ricordare e' che le moltiplicazioni e le divisioni sono LENTISSIME, per cui toglierle di mezzo sostituendole con qualcos'altro e' molto utile.

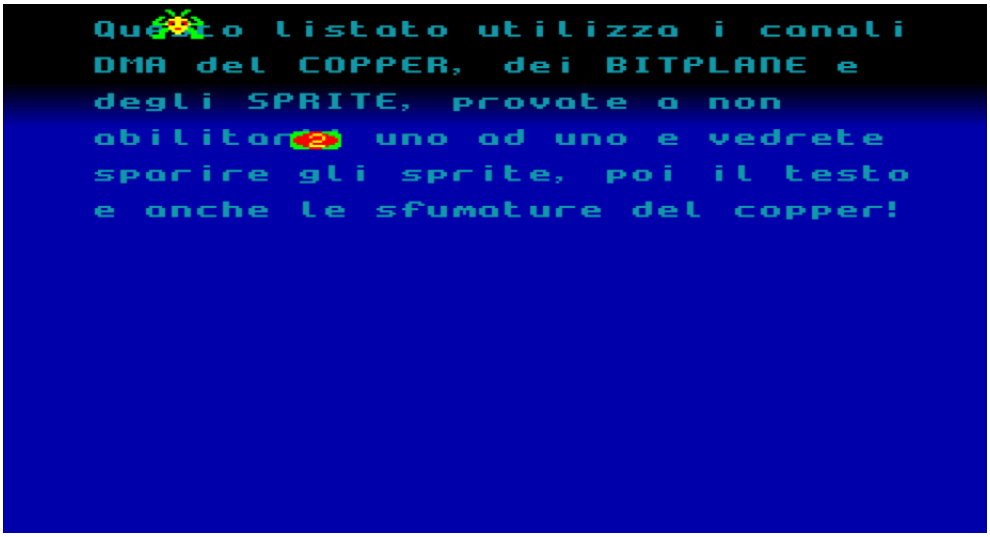


Figura 24.2: Lezione 8b

24.3 Lezione8c

```
; Una routine di FADE (ossia dissolvenza) da e verso il NERO. ROUTINE N.1
; Premere il tasto sinistro e destro

        SECTION       Fade1,CODE
```

```
;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA
;                 -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
;        puntiamo la figura

        MOVE.L        #Logo1,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        MOVEQ        #4-1,D1                ; numero di bitplanes (qua sono 4)
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*84,d0        ; + lunghezza bitplane (qua e' alto 84 linee)
        addq.w        #8,a1
        dbra        d1,POINTBP


        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA


mouse1:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse1

        clr.w        FaseDelFade        ; azzera il numero del fotogramma

;        ********** primo fade: dal NERO ai colori *********

mouse2:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse2
Aspetta1:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta1

        bsr.w        FadeIN                ; Fade!!!

        btst        #2,$dff016        ; mouse premuto?
```

```
        bne.s        mouse2

        move.w        #16,FaseDelFade        ; azzera il numero del fotogramma

;        ********** secondo fade: dai colori al NERO *********

mouse3:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse3
Aspetta2:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta2

        bsr.w        FadeOUT        ; Fade!!!

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse3
        rts


*****************************************************************************
;        Routines che aspettano e richiamano Fade al momento giusto
*****************************************************************************

FadeIn:
        cmp.w        #17,FaseDelFade
        beq.s        FinitoFadeIn
        moveq        #0,d0
        move.w        FaseDelFade(PC),d0
        moveq        #15-1,d7                ; D7 = Numero di colori
        lea        TabColoriPic(PC),a0        ; A0 = indirizzo tabella dei colori
                                        ; della figura da "dissolvere"
        lea        CopColors+6,a1                ; A1 = indirizzo colori in copperlist
                                        ; da notare che parte dal COLOR1 e
                                        ; non dal color0, in quanto il color0
                                        ; e'=$000 e cosi' rimane.
        bsr.s        Fade
        addq.w        #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
FinitoFadeIn:
        rts


FadeOut:
        tst.w        FaseDelFade        ; abbiamo superato l'ultima fase? (16)?
        beq.s        FinitoOut
        subq.w        #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
        moveq        #0,d0
        move.w        FaseDelFade(PC),d0
        moveq        #15-1,d7                ; D7 = Numero di colori
        lea        TabColoriPic(PC),a0        ; A0 = indirizzo tabella dei colori
                                        ; della figura da "dissolvere"
        lea        CopColors+6,a1                ; A1 = indirizzo colori in copperlist
                                        ; da notare che parte dal COLOR1 e
                                        ; non dal color0, in quanto il color0
                                        ; e'=$000 e cosi' rimane.
        bsr.s        Fade
FinitoOut:
        rts

FaseDelFade:                ; fase attuale del fade (0-16)
        dc.w        0
```

```
*****************************************************************************
*               Routine per Fade In/Out da e verso il NERO                  *
* Input:                                                                   *
*                                                                           *
* d7 = Numero colori-1                                                        *
* a0 = Indirizzo tabella con i colori della figura                          *
* a1 = Indirizzo primo colore in copperlist                                 *
* d0 = Momento del fade, multiplier - per esempio con d0=0 lo schermo         *
*        e' nero totalmente, con d0=8 siamo a meta' fade e con d0=16         *
*        siamo ai colori pieni; dunque ci sono 17 fasi, dalla 0 alla 16.        *
*        Per fare un fade IN, dal nero al colore, si deve dare a ogni         *
*        chiamata alla routine un valore di d0 crescente da 0 a 16          *
*        Per un fade OUT, si dovra' partire da d0=16 fino a d0=0                *
*                                                                           *
*  Il procedimento di FADE e' quello di moltiplicare ogni componente R,G,B  *
*  del colore per un Multiplier, che va da 0 per il NERO (x*0=0), a 16 per  *
*  i colori normali, dato che poi il colore viene diviso per 16,            *
*  moltiplicare un colore per 16 e ridividerlo non fa che lasciarlo uguale. *
*                                                                           *
*****************************************************************************


;               \   /
;               .\-/.
;           /\ ()   ()
;          /  \/~---~\.-~^-.
;        .-~^-./   |   \---.
;             {    |    }   \
;           .-~\   |   /~-.
;          /    \  I  /    \
;               \/ \/

Fade:
ColorLoop:
        moveq        #0,d1               ; azzera D1
        moveq        #0,d2               ; azzera D2

; Trova la componente risultante ROSSA (Red) ed inseriscila in copperlist ($0R)

        move.b        (a0)+,d1        ; D1.b = componente RED (ROSSA) del colore
                                  ; ossia $0R (la word e' $0RGB)
        mulu.w        d0,d1                ; Moltiplicalo per il livello colore attuale
        lsr.w        #4,d1                ; Dividilo per 16 (con LSR #4), portando il
                                  ; risultato a destra (il byte e' %00001111)
        move.b        d1,(a1)+        ; Inserisci la nuova componente ROSSA in
                                  ; copperlist (ossia il byte $0R)

; Trova la componente risultante VERDE (Green) e mettila in d1

        move.b        (a0),d1                ; D1.b = componente Green,Blue (VERDE,BLU)
                                  ; ossia $GB (la word e' $0RGB)
        lsr.b        #4,d1                ; Metti il VERDE tutto a destra spostando
                                  ; il valore a destra di 4 bit (1 nibble)
                                  ; dunque in d1.b abbiamo solo il verde
        mulu.w        d0,d1                ; Moltiplicalo per il livello colore attuale
        and.b        #$f0,d1                ; Mascheriamo per selezionare solo il risultato
                                  ; che a questo punto e' pronto, non occorre
                                  ; spostarlo a destra, dato che nel registro
                                  ; colore si trova proprio in questa posizione.
                                  ; infatti il byte basso e' $GB (la word $0RGB)

;  Trova la componente risultante BLU e mettila in d2
```

```
        move.b        (a0)+,d2        ; D2.b = componente Green,Blue (VERDE,BLU)
                                ; ossia $GB (la word e' $ORGB)
        and.b        #$0f,d2                ; Mascheriamo per selezionare solo il BLU ($0B)
        mulu.w        d0,d2                ; Moltiplicalo per il livello colore attuale
        lsr.w        #4,d2                ; Dividi per 16, portando il risultato a destra
                                ; in modo che il risultato sia $0B

; Unisci con OR la componente risultante VERDE con quella BLU

        or.w        d2,d1                ; OR del BLU con il VERDE per "unirli" nel byte
                                ; finale risultante: $GB

; E metti il byte risultante $GB in copperlist

        move.b        d1,(a1)+        ; Inserisci il valore del VERDE e del BLU nel
                                ; byte basso $GB del colore in copperlist
        addq.w        #2,a1                ; Vai al prossimo colore i copperlist, saltando
                                ; la word del $18x
        dbra        d7,ColorLoop        ; ripeti per gli altri colori
        rts


; il $180, color0, e' $000, dunque non cambia! La tabella parte dal color1

TabColoriPic:
        dc.w $fff,$200,$310,$410,$620,$841,$a73
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446


*****************************************************************************
;                        Copper List
*****************************************************************************
        section        copper,data_c                ; Chip data

Copperlist:
        dc.w        $8E,$2c81        ; DiwStrt - window start
        dc.w        $90,$2cc1        ; DiwStop - window stop
        dc.w        $92,$38                ; DdfStart - data fetch start
        dc.w        $94,$d0                ; DdfStop - data fetch stop
        dc.w        $102,0                ; BplCon1 - scroll register
        dc.w        $104,0                ; BplCon2 - priority register
        dc.w        $108,0                ; Bpl1Mod - modulo pl. dispari
        dc.w        $10a,0                ; Bpl2Mod - modulo pl. pari

                    ; 5432109876543210
        dc.w        $100,%0100001000000000        ; BPLCON0 - 4 planes lowres (16 colori)

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane

; i primi 16 colori sono per il LOGO

CopColors:
        dc.w $180,0,$182,0,$184,0,$186,0
        dc.w $188,0,$18a,0,$18c,0,$18e,0
        dc.w $190,0,$192,0,$194,0,$196,0
```

```
        dc.w $198,0,$19a,0,$19c,0,$19e,0

;        dc.w $180,$000,$182,$fff,$184,$200,$186,$310
;        dc.w $188,$410,$18a,$620,$18c,$841,$18e,$a73
;        dc.w $190,$b95,$192,$db6,$194,$dc7,$196,$111
;        dc.w $198,$222,$19a,$334,$19c,$99b,$19e,$446

;        Mettiamo un poco di sfumature per la scenografia...

        dc.w        $8007,$fffe        ; Wait - $2c+84=$80
        dc.w        $100,$200        ; bplcon0 - no bitplanes
        dc.w        $180,$003        ; color0
        dc.w        $8207,$fffe        ; wait
        dc.w        $180,$005        ; color0
        dc.w        $8507,$fffe        ; wait
        dc.w        $180,$007        ; color0
        dc.w        $8a07,$fffe        ; wait
        dc.w        $180,$009        ; color0
        dc.w        $9207,$fffe        ; wait
        dc.w        $180,$00b        ; color0

        dc.w        $9e07,$fffe        ; wait
        dc.w        $180,$999        ; color0
        dc.w        $a007,$fffe        ; wait
        dc.w        $180,$666        ; color0
        dc.w        $a207,$fffe        ; wait
        dc.w        $180,$222        ; color0
        dc.w        $a407,$fffe        ; wait
        dc.w        $180,$001        ; color0

        dc.l        $ffff,$fffe        ; Fine della copperlist


*****************************************************************************
;                                DISEGNO
*****************************************************************************

        section        gfxstuff,data_c

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).

; Il logo e' copyright FLENDER/RAM JAM

Logo1:
        incbin        'logo320*84*16c.raw'

        end
```

Questo listato offre la visione di un FADE, ossia di una dissolvenza dal NERO al colore, e dal colore al NERO. Dato che la routine FADE richiede di essere richiamata 16 volte per trasformare i colori dal NERO a quelli finali, e altre 16 volte per tornare al nero dai colori, e' stato necessario scrivere 2 routines ausiliarie, FadeIn e FadeOut, che richiamano la routine Fade, quella vera e propria, passandogli un valore del Multiplier diverso ogni volta, salvata nella label FaseDelFade.
Il procedimento di FADE e' quello di moltiplicare ogni componente R,G,B del colore per un Multiplier, che va da 0 per il NERO (x*0=0), a 16 per i colori normali, dato che poi il colore viene diviso per 16, moltiplicare un colore per 16 e ridividerlo non fa che lasciarlo invariato.
La routine di questo esempio e' la NUMERO 1, e lavora separatamente sui due bytes della word colore. Il prossimo listato contiene una routine che usa lo stesso procedimento della moltiplicazione per il multiplier e della

```
divisione per 16, ma non aggiorna la word colore un byte alla volta, e forse
puo' risultarvi piu' chiara. Comunque, o capite questo esempio o il prossimo!
```

24.4 Lezione8d

```
; Una routine di FADE (ossia dissolvenza) da e verso il NERO. ROUTINE N.2
; Premere il tasto sinistro e destro

        SECTION        Fade1,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
***************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA
;                 -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
;        puntiamo la figura

        MOVE.L        #Logo1,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        MOVEQ        #4-1,D1                ; numero di bitplanes (qua sono 4)
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*84,d0        ; + lunghezza bitplane (qua e' alto 84 linee)
        addq.w        #8,a1
        dbra        d1,POINTBP


        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA


mouse1:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse1

        clr.w        FaseDelFade        ; azzera il numero del fotogramma
```

```
mouse2:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse2
Aspetta1:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta1

        bsr.w        FadeIN                ; Fade!!!

        btst        #2,$dff016        ; mouse premuto?
        bne.s        mouse2

        move.w        #16,FaseDelFade        ; parti dal fotogramma 16

mouse3:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse3
Aspetta2:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta2

        bsr.w        FadeOUT        ; Fade!!!

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse3

        rts


*****************************************************************************
;        Routines che aspettano e richiamano Fade al momento giusto
*****************************************************************************

FadeIn:
        cmp.w        #17,FaseDelFade
        beq.s        FinitoFadeIn
        moveq        #0,d0
        move.w        FaseDelFade(PC),d0
        moveq        #15-1,d7                ; D7 = Numero di colori
        lea        TabColoriPic(PC),a0        ; A0 = indirizzo tabella dei colori
                                        ; della figura da "dissolvere"
        lea        CopColors+6,a1                ; A1 = indirizzo colori in copperlist
                                        ; da notare che parte dal COLOR1 e
                                        ; non dal color0, in quanto il color0
                                        ; e'=$000 e cosi' rimane.
        bsr.s        Fade
        addq.w        #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
FinitoFadeIn:
        rts


FadeOut:
        tst.w        FaseDelFade        ; abbiamo superato l'ultima fase? (16)?
        beq.s        FinitoOut
        subq.w        #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
        moveq        #0,d0
        move.w        FaseDelFade(PC),d0
        moveq        #15-1,d7                ; D7 = Numero di colori
        lea        TabColoriPic(PC),a0        ; A0 = indirizzo tabella dei colori
                                        ; della figura da "dissolvere"
        lea        CopColors+6,a1                ; A1 = indirizzo colori in copperlist
                                        ; da notare che parte dal COLOR1 e
```

```
                                        ; non dal color0, in quanto il color0
                                        ; e'=$000 e cosi' rimane.
        bsr.s        Fade
FinitoOut:
        rts

FaseDelFade:                ; fase attuale del fade (0-16)
        dc.w        0

*****************************************************************************
*               Routine per Fade In/Out da e verso il NERO (versione 2)           *
* Input:                                                                  *
*                                                                          *
* d7 = Numero colori-1                                                          *
* a0 = Indirizzo tabella con i colori della figura                          *
* a1 = Indirizzo primo colore in copperlist                                  *
* d0 = Momento del fade, multiplier - per esempio con d0=0 lo schermo            *
*        e' bianco totalmente, con d0=8 siamo a meta' fade e con d0=16            *
*        siamo ai colori pieni; dunque ci sono 17 fasi, dalla 0 alla 16.            *
*        Per fare un fade IN, dal bianco al colore, si deve dare a ogni            *
*        chiamata alla routine un valore di d0 crescente da 0 a 16            *
*        Per un fade OUT, si dovra' partire da d0=16 fino a d0=0                  *
*                                                                          *
*  Il procedimento di FADE e' quello di moltiplicare ogni componente R,G,B  *
*  del colore per un Multiplier, che va da 0 per il NERO (x*0=0), a 16 per  *
*  i colori normali, dato che poi il colore viene diviso per 16,            *
*  moltiplicare un colore per 16 e ridividerlo non fa che lasciarlo uguale. *
*                                                                          *
*****************************************************************************

;           .      .-~\
;          / '-'\.'    '- :
;          |    /          '._
;          |   |   .-.        {
;           \  |   '-'         '.
;         .  \ |                /
;        ~-.'. \|             .-~_
;          '.\-.\       .-~      \
;            '-'/~~ -.~           /
;          .-~/|'-._ /~~-.~ -- ~
;         /  |  \    ~- . _\

Fade:
;        Calcola la componente BLU

        MOVE.W        (A0),D4                ; Metti il colore dalla tabella colori in d4
        AND.W        #$00f,D4        ; Seleziona solo la componente blu ($RGB->$00B)
        MULU.W        D0,D4                ; Moltiplica per la fase del fade (0-16)
        ASR.W        #4,D4                ; shift 4 BITS a destra, ossia divisione per 16
        AND.W        #$00f,D4        ; Seleziona solo la componente BLU
        MOVE.W        D4,D5                ; Salva la componente BLU in d5

;        Calcola la componente VERDE (GREEN)

        MOVE.W        (A0),D4                ; Metti il colore dalla tabella colori in d4
        AND.W        #$0f0,D4        ; Selez. solo la componente verde ($RGB->$0G0)
        MULU.W        D0,D4                ; Moltiplica per la fase del fade (0-16)
        ASR.W        #4,D4                ; shift 4 BITS a destra, ossia divisione per 16
        AND.W        #$0f0,D4        ; Seleziona solo la componente VERDE
        OR.W        D4,D5                ; Salva la comp.verde assieme a quella BLU

;        Calcola la componente ROSSA (RED)
```

```
        MOVE.W        (A0)+,D4        ; leggi il colore dalla tabella
                                   ; e fai puntare a0 al prossimo col.
        AND.W        #$f00,D4        ; Selez. solo la componente rossa ($RGB->$R00)
        MULU.W        D0,D4               ; Moltiplica per la fase del fade (0-16)
        ASR.W        #4,D4               ; shift 4 BITS a destra, ossia divisione per 16
        AND.W        #$f00,D4        ; Selez. solo la componente rossa ($RGB->$R00)
        OR.W        D4,D5                ; Salva la c. ROSSA assieme alla BLU e VERDE

        MOVE.W        D5,(A1)               ; E metti il colore $0RGB finale in copperlist
        addq.w        #4,a1               ; prossimo colore in copperlist
        DBRA        D7,Fade               ; fai tutti i colori
        rts


; il $180, color0, e' $000, dunque non cambia! La tabella parte dal color1

TabColoriPic:
        dc.w $fff,$200,$310,$410,$620,$841,$a73
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446


*****************************************************************************
;                          Copper List
*****************************************************************************
        section        copper,data_c                ; Chip data

Copperlist:
        dc.w        $8E,$2c81        ; DiwStrt - window start
        dc.w        $90,$2cc1        ; DiwStop - window stop
        dc.w        $92,$38               ; DdfStart - data fetch start
        dc.w        $94,$d0               ; DdfStop - data fetch stop
        dc.w        $102,0               ; BplCon1 - scroll register
        dc.w        $104,0               ; BplCon2 - priority register
        dc.w        $108,0               ; Bpl1Mod - modulo pl. dispari
        dc.w        $10a,0               ; Bpl2Mod - modulo pl. pari

                    ; 5432109876543210
        dc.w        $100,%0100001000000000        ; BPLCON0 - 4 planes lowres (16 colori)

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane

; i primi 16 colori sono per il LOGO

CopColors:
        dc.w $180,0,$182,0,$184,0,$186,0
        dc.w $188,0,$18a,0,$18c,0,$18e,0
        dc.w $190,0,$192,0,$194,0,$196,0
        dc.w $198,0,$19a,0,$19c,0,$19e,0

;        dc.w $180,$000,$182,$fff,$184,$200,$186,$310
;        dc.w $188,$410,$18a,$620,$18c,$841,$18e,$a73
;        dc.w $190,$b95,$192,$db6,$194,$dc7,$196,$111
;        dc.w $198,$222,$19a,$334,$19c,$99b,$19e,$446

;        Mettiamo un poco di sfumature per la scenografia...
```

```
        dc.w        $8007,$fffe        ; Wait - $2c+84=$80
        dc.w        $100,$200        ; bplcon0 - no bitplanes
        dc.w        $180,$003        ; color0
        dc.w        $8207,$fffe        ; wait
        dc.w        $180,$005        ; color0
        dc.w        $8507,$fffe        ; wait
        dc.w        $180,$007        ; color0
        dc.w        $8a07,$fffe        ; wait
        dc.w        $180,$009        ; color0
        dc.w        $9207,$fffe        ; wait
        dc.w        $180,$00b        ; color0

        dc.w        $9e07,$fffe        ; wait
        dc.w        $180,$999        ; color0
        dc.w        $a007,$fffe        ; wait
        dc.w        $180,$666        ; color0
        dc.w        $a207,$fffe        ; wait
        dc.w        $180,$222        ; color0
        dc.w        $a407,$fffe        ; wait
        dc.w        $180,$001        ; color0

        dc.l        $ffff,$fffe        ; Fine della copperlist


*****************************************************************************
;                                DISEGNO
*****************************************************************************

        section        gfxstuff,data_c

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).

Logo1:
        incbin        'logo320*84*16c.raw'

        end
```

Questa routine funziona esattamente come la precedente, ma non scrive la
word colore un byte alla volta. Questa routine si presta di piu' alle modifiche
per farla divenire AGA. Infatti la vedremo "AGhizzAta" nella lezione sull'AGA.
Il discorso della moltiplicazione e della divisione delle componenti R,G,B
sara' ampliato ad un byte per ogni componente R,G,B.

24.5 Lezione8e

```
; Una routine di FADE (ossia dissolvenza) con una modifica riguardante il mix
; dei colori, infatti si puo' indicare una sfumatura RGB verso la quale
; tendere. Premere il tasto sinistro e destro varie volte per vedere e uscire.

        SECTION        Fade1,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA
```

```
;                -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
;        puntiamo la figura

        MOVE.L        #Logo1,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        MOVEQ        #4-1,D1               ; numero di bitplanes (qua sono 4)
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*84,d0        ; + lunghezza bitplane (qua e' alto 84 linee)
        addq.w        #8,a1
        dbra        d1,POINTBP


        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                    ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA


mouse1:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse1

;        ****************+ parte 1 con tendenza al ROSSO *****************

        clr.w        FaseDelFade        ; azzera il numero del fotogramma
        move.w        #$b12,Tendenza        ; imposta la tendenza al rosso ******

mouse2:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse2
Aspetta1:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta1

        bsr.w        FadeIN                ; Fade!!!

        btst        #2,$dff016        ; mouse premuto?
        bne.s        mouse2

        move.w        #16,FaseDelFade        ; parti dal fotogramma 16

mouse3:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse3
```

```
Aspetta2:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta2

        bsr.w        FadeOUT        ; Fade!!!

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse3

;        ****************+ parte 2 con tendenza al VERDE *****************

        clr.w        FaseDelFade        ; azzera il numero del fotogramma
        move.w        #$373,Tendenza        ; imposta la tendenza al verde ******

mouse2x:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse2x
Aspetta1x:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta1x

        bsr.w        FadeIN                ; Fade!!!

        btst        #2,$dff016        ; mouse premuto?
        bne.s        mouse2x

        move.w        #16,FaseDelFade        ; parti dal fotogramma 16

mouse3x:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse3x
Aspetta2x:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta2x

        bsr.w        FadeOUT        ; Fade!!!

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse3x


;        ****************+ parte 3 con tendenza al BLU *****************

        clr.w        FaseDelFade        ; azzera il numero del fotogramma
        move.w        #$33c,Tendenza        ; imposta la tendenza al BLU ******

mouse2y:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse2y
Aspetta1y:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta1y

        bsr.w        FadeIN                ; Fade!!!

        btst        #2,$dff016        ; mouse premuto?
        bne.s        mouse2y

        move.w        #16,FaseDelFade        ; parti dal fotogramma 16

mouse3y:
        CMP.b        #$ff,$dff006        ; linea 255
```

```
        bne.s        mouse3y
Aspetta2y:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta2y

        bsr.w        FadeOUT        ; Fade!!!

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse3y

        rts


*****************************************************************************
;        Routines che aspettano e richiamano Fade al momento giusto
*****************************************************************************

FadeIn:
        cmp.w        #17,FaseDelFade
        beq.s        FinitoFadeIn
        moveq        #0,d0
        move.w        FaseDelFade(PC),d0
        moveq        #15-1,d7                ; D7 = Numero di colori
        lea        TabColoriPic(PC),a0        ; A0 = indirizzo tabella dei colori
                                        ; della figura da "dissolvere"
        lea        CopColors+6,a1                ; A1 = indirizzo colori in copperlist
                                        ; da notare che parte dal COLOR1 e
                                        ; non dal color0, in quanto il color0
                                        ; e'=$000 e cosi' rimane.
        bsr.s        Fade
        addq.w        #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
FinitoFadeIn:
        rts


FadeOut:
        tst.w        FaseDelFade        ; abbiamo superato l'ultima fase? (16)?
        beq.s        FinitoOut
        subq.w        #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
        moveq        #0,d0
        move.w        FaseDelFade(PC),d0
        moveq        #15-1,d7                ; D7 = Numero di colori
        lea        TabColoriPic(PC),a0        ; A0 = indirizzo tabella dei colori
                                        ; della figura da "dissolvere"
        lea        CopColors+6,a1                ; A1 = indirizzo colori in copperlist
                                        ; da notare che parte dal COLOR1 e
                                        ; non dal color0, in quanto il color0
                                        ; e'=$000 e cosi' rimane.
        bsr.s        Fade
FinitoOut:
        rts

FaseDelFade:                ; fase attuale del fade (0-16)
        dc.w        0

*****************************************************************************
*                Routine per Fade In/Out da e verso il BIANCO                    *
* Input:                                                                    *
*                                                                             *
* d7 = Numero colori-1                                                            *
* a0 = Indirizzo tabella con i colori della figura                            *
* a1 = Indirizzo primo colore in copperlist                                    *
```

```
* d0 = Momento del fade, multiplier - per esempio con d0=0 lo schermo            *
*        e' bianco totalmente, con d0=8 siamo a meta' fade e con d0=16            *
*        siamo ai colori pieni; dunque ci sono 17 fasi, dalla 0 alla 16.            *
*        Per fare un fade IN, dal bianco al colore, si deve dare a ogni            *
*        chiamata alla routine un valore di d0 crescente da 0 a 16           *
*        Per un fade OUT, si dovra' partire da d0=16 fino a d0=0                    *
* d6 = colore da aggiungere al fade per dargli certe tonalita'                     *
*                                                                          *
*  Il procedimento di FADE e' quello di moltiplicare ogni componente R,G,B  *
*  del colore per un Multiplier, che va da 0 per il NERO (x*0=0), a 16 per  *
*  i colori normali, dato che poi il colore viene diviso per 16,            *
*  moltiplicare un colore per 16 e ridividerlo non fa che lasciarlo uguale. *
*                                                                           *
* La modifica consiste semplicemente nell'aggiungere le tonalita' in d6     *
* e dividere il risultato per 2, per evitare di superare un totale di $f    *
*                                                                           *
*****************************************************************************

;                       .-~~~-.
;                      /        }
;                     /      .-~
;          \         |        }
;        ___\.~~-.-~|     .-~_
;           { O  |  '  -~       ~-._
;            ~--~/-|_\          .-~
;               /  |  \~ - - ~
;              /   |   \

Fade:
        MOVE.W        Tendenza(PC),D6        ;        ; Maschera di tendenza RGB
Fade1:
        MOVE.W        D6,D1                ; copia col. tendenz. in d1
        MOVE.W        D6,D2                ; in d2
        MOVE.W        D6,D3                ; in d3
        ANDI.W        #$00f,D1        ; SELEZ. SOLO BLUE
        ANDI.W        #$0f0,D2        ; SELEZ. SOLO GREEN
        ANDI.W        #$f00,D3        ; SELEZ. SOLO RED

;        Calcola la componente BLU

        MOVE.W        (A0),D4                ; Metti il colore dalla tabella colori in d4
        AND.W        #$00f,D4        ; Seleziona solo la componente blu ($RGB->$00B)
; modifica
        ADD.W        D1,D4                ; Aggiungi la componente BLU di Tendenza
        LSR.W        #1,D4                ; e dividi per 2 tramite uno shift di 1 bit a >
; fine modifica
        MULU.W        D0,D4                ; Moltiplica per la fase del fade (0-16)
        ASR.W        #4,D4                ; shift 4 BITS a destra, ossia divisione per 16
        AND.W        #$00f,D4        ; Seleziona solo la componente BLU
        MOVE.W        D4,D5                ; Salva la componente BLU in d5

;        Calcola la componente VERDE (GREEN)

        MOVE.W        (A0),D4                ; Metti il colore dalla tabella colori in d4
        AND.W        #$0f0,D4        ; Selez. solo la componente verde ($RGB->$0G0)
; modifica
        ADD.W        D2,D4                ; Aggiungi la componente VERDE di Tendenza
        LSR.W        #1,D4                ; e dividi per 2 tramite uno shift di 1 bit a >
; fine modifica
        MULU.W        D0,D4                ; Moltiplica per la fase del fade (0-16)
        ASR.W        #4,D4                ; shift 4 BITS a destra, ossia divisione per 16
        AND.W        #$0f0,D4        ; Seleziona solo la componente VERDE
```

```
        OR.W        D4,D5               ; Salva la comp.verde assieme a quella BLU


;        Calcola la componente ROSSA (RED)

        MOVE.W        (A0)+,D4        ; leggi il colore dalla tabella
                                ; e fai puntare a0 al prossimo col.
        AND.W        #$f00,D4        ; Selez. solo la componente rossa ($RGB->$R00)
; modifica
        ADD.W        D3,D4                ; Aggiungi la componente ROSSA di Tendenza
        LSR.W        #1,D4                ; e dividi per 2 tramite uno shift di 1 bit a >
; fine modifica
        MULU.W        D0,D4                ; Moltiplica per la fase del fade (0-16)
        ASR.W        #4,D4                ; shift 4 BITS a destra, ossia divisione per 16
        AND.W        #$f00,D4        ; Selez. solo la componente rossa ($RGB->$R00)
        OR.W        D4,D5                ; Salva la c. ROSSA assieme alla BLU e VERDE

        MOVE.W        D5,(A1)                ; E metti il colore $0RGB finale in copperlist
        addq.w        #4,a1                ; prossimo colore in copperlist
        DBRA        D7,Fade1        ; fai tutti i colori
        rts


Tendenza:
        dc.w        0

; il $180, color0, e' $000, dunque non cambia! La tabella parte dal color1

TabColoriPic:
        dc.w $fff,$200,$310,$410,$620,$841,$a73
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446


*****************************************************************************
;                        Copper List
*****************************************************************************
        section        copper,data_c                ; Chip data

Copperlist:
        dc.w        $8E,$2c81        ; DiwStrt - window start
        dc.w        $90,$2cc1        ; DiwStop - window stop
        dc.w        $92,$38                ; DdfStart - data fetch start
        dc.w        $94,$d0                ; DdfStop - data fetch stop
        dc.w        $102,0                ; BplCon1 - scroll register
        dc.w        $104,0                ; BplCon2 - priority register
        dc.w        $108,0                ; Bpl1Mod - modulo pl. dispari
        dc.w        $10a,0                ; Bpl2Mod - modulo pl. pari

                ; 5432109876543210
        dc.w        $100,%0100001000000000        ; BPLCON0 - 4 planes lowres (16 colori)

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane

; i primi 16 colori sono per il LOGO

CopColors:
```

```
        dc.w $180,0,$182,0,$184,0,$186,0
        dc.w $188,0,$18a,0,$18c,0,$18e,0
        dc.w $190,0,$192,0,$194,0,$196,0
        dc.w $198,0,$19a,0,$19c,0,$19e,0

;        dc.w $180,$000,$182,$fff,$184,$200,$186,$310
;        dc.w $188,$410,$18a,$620,$18c,$841,$18e,$a73
;        dc.w $190,$b95,$192,$db6,$194,$dc7,$196,$111
;        dc.w $198,$222,$19a,$334,$19c,$99b,$19e,$446

;        Mettiamo un poco di sfumature per la scenografia...

        dc.w        $8007,$fffe        ; Wait - $2c+84=$80
        dc.w        $100,$200        ; bplcon0 - no bitplanes
        dc.w        $180,$003        ; color0
        dc.w        $8207,$fffe        ; wait
        dc.w        $180,$005        ; color0
        dc.w        $8507,$fffe        ; wait
        dc.w        $180,$007        ; color0
        dc.w        $8a07,$fffe        ; wait
        dc.w        $180,$009        ; color0
        dc.w        $9207,$fffe        ; wait
        dc.w        $180,$00b        ; color0

        dc.w        $9e07,$fffe        ; wait
        dc.w        $180,$999        ; color0
        dc.w        $a007,$fffe        ; wait
        dc.w        $180,$666        ; color0
        dc.w        $a207,$fffe        ; wait
        dc.w        $180,$222        ; color0
        dc.w        $a407,$fffe        ; wait
        dc.w        $180,$001        ; color0

        dc.l        $ffff,$fffe        ; Fine della copperlist


*****************************************************************************
;                                DISEGNO
*****************************************************************************

        section        gfxstuff,data_c

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).

Logo1:
        incbin        'logo320*84*16c.raw'

        end
```

Questa e' una semplice modifica della routine precedente.

24.6 Lezione8f

```
; Una routine di FADE (ossia dissolvenza) da e verso il QUALSIASI COLORE
; Premere il tasto sinistro e destro alternativamente per vedere i vari
; utilizzi della routine e per uscire

        SECTION        Fade1,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include         "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA
;                  -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
;        puntiamo la figura

        MOVE.L        #Logo1,d0        ; dove puntare
        LEA         BPLPOINTERS,A1        ; puntatori COP
        MOVEQ         #4-1,D1                ; numero di bitplanes (qua sono 4)
POINTBP:
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)
        swap         d0
        ADD.L         #40*84,d0        ; + lunghezza bitplane (qua e' alto 84 linee)
        addq.w        #8,a1
        dbra         d1,POINTBP


        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

mouse1:
        btst         #6,$bfe001        ; mouse premuto?
        bne.s         mouse1

;        ********** primo fade: dal NERO ai colori *********

mouse2:
        CMP.b         #$ff,$dff006        ; linea 255
        bne.s         mouse2
Aspetta1:
        CMP.b         #$ff,$dff006        ; linea 255
        beq.s         Aspetta1

        lea         Cstart1-2,a1        ; Start-colour-table
        lea         Cend1-2,a2        ; End-colour-table
        bsr.w         dofade                ; Fade!!!


        btst         #2,$dff016        ; mouse premuto?
        bne.s         mouse2
```

```
        clr.w       FaseDelFade               ; azzera il numero del fotogramma

;        ********** secondo fade: dai colori al NERO *********

mouse3:
        CMP.b       #$ff,$dff006        ; linea 255
        bne.s       mouse3
Aspetta2:
        CMP.b       #$ff,$dff006        ; linea 255
        beq.s       Aspetta2

        lea        Cstart2-2,a1        ; Start-colour-table
        lea        Cend2-2,a2        ; End-colour-table
        bsr.w        dofade               ; Fade!!!


        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse3

        clr.w       FaseDelFade               ; azzera il numero del fotogramma

;        ********** terzo fade: dal BIANCO ai colori *********

mouse4:
        CMP.b       #$ff,$dff006        ; linea 255
        bne.s       mouse4
Aspetta3:
        CMP.b       #$ff,$dff006        ; linea 255
        beq.s       Aspetta3

        lea        Cstart3-2,a1        ; Start-colour-table
        lea        Cend3-2,a2        ; End-colour-table
        bsr.w        dofade               ; Fade!!!


        btst        #2,$dff016        ; mouse premuto?
        bne.s        mouse4

        clr.w       FaseDelFade               ; azzera il numero del fotogramma

;        ********** quarto fade: dai COLORI ad altri colori diversi! *********

mouse5:
        CMP.b       #$ff,$dff006        ; linea 255
        bne.s       mouse5
Aspetta4:
        CMP.b       #$ff,$dff006        ; linea 255
        beq.s       Aspetta4

        lea        Cstart4-2,a1        ; Start-colour-table
        lea        Cend4-2,a2        ; End-colour-table
        bsr.w        dofade               ; Fade!!!


        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse5

        clr.w       FaseDelFade               ; azzera il numero del fotogramma

;        ********** quinto fade: dai COLORI ad altri colori diversi! *********
```

```
mouse6:
        CMP.b        #$ff,$dff006        ; linea 255
        bne.s        mouse6
Aspetta5:
        CMP.b        #$ff,$dff006        ; linea 255
        beq.s        Aspetta5

        lea        Cstart5-2,a1        ; Start-colour-table
        lea        Cend5-2,a2        ; End-colour-table
        bsr.w        dofade                ; Fade!!!

        btst        #2,$dff016        ; mouse premuto?
        bne.s        mouse6

        rts


*****************************************************************************
*        Routine per Fade In/Out da e verso qualsiasi colore!                    *
* Input:                                                                    *
*                                                                            *
* d6 = Numero colori-1                                                          *
* a1 = Indirizzo tabella1 con i colori della figura (sorgente)                  *
* a2 = Indirizzo tabella2 con i colori della figura (destinazione)            *
* a0 = Indirizzo primo colore in copperlist                                   *
* label FaseDelFade usata come il d0 per le routines precedenti,            *
*   = Momento del fade, multiplier, in questo caso pero' occorre azzerarlo  *
*     semplicemente per far partire un nuovo fade                             *
*                                                                            *
*  Il funzionamento della routine e' piu' complesso delle precedenti, e     *
*  per la verita' non mi ricordo neppure io come funziona esattamente.      *
*  Leggete i miei vecchi commenti, comunque basta che sappiate come usarla  *
*                                                                            *
*****************************************************************************

;                   .--._.--.
;                  ( O     O )
;                  /   . .   \
;                 .‘.-------.’.
;                /(     \_/    )\
;              _/  \  \   /  /  \_
;           .~   ‘  \  \ /  /  ’   ~.
;          {    -.   \  V  /   .-    }
;        _ _‘.    \  |  |  |  /    .’_ _
;        >_       _} |  |  | {_       _<
;         /. - ~ ,_-’  .^.  ‘-_, ~ - .\
;                 ’-’|/   \|‘-‘

dofade:
        cmp.w        #17,FaseDelFade        ; abbiamo superato l’ultima fase? (16)?
        beq.s        FadeFinito
        lea        CopColors+2,a0        ; Copper
        move.w        #15-1,d6        ; No.  colours
        bsr.w        fade2                ; Do fading!
FadeFinito:
        rts

; Uses d0-d6/a0-a2

fade2:
f2main:
        addq.w        #4,a0        ; vai al prossimo registro colore in copperlist
```

```
        addq.w      #2,a1       ; vai al prossimo colore della tabella colori sorg.
        addq.w      #2,a2       ; idem per la tabella colori destinaz.
        move.w      (a0),d0             ; colore dalla copperlist in d0
        move.w      (a2),d1             ; col. tab destinazione in d1
        cmp.w       d0,d1               ; sono uguali?
        beq.w       ProssimoColore              ; se si, vai al prossimo colore
        move.w      FaseDelFade(PC),d4      ; fase del fade in d4 (0-16)
        clr.w       ColoreFinale             ; azzera il colore finale

;       BLU

        move.w      (a1),d0             ; colore attuale della tab sorgente in d0
        move.w      (a2),d2             ; colore tab destinazione in d2
        and.l       #$00f,d0        ; selez. solo il BLU dal colore tab sorgente
        and.l       #$00f,d2        ; idem per colore tab destinazione
        cmp.w       d2,d0               ; i BLU sorg. e destinazione sono uguali?
        bhi.b       SottraiD2       ; se d2>d0, FadCh1a
        beq.b       SottraiD2       ; se sono uguali, Sottrai d2
        sub.w       d0,d2               ; se d2<d0, subba d0 a d2!
        bra.b       SottFatto
SottraiD2:
        sub.w       d2,d0               ; altrimenti subba d2 a d0!
        bra.b       SottFatto2

SottFatto:
        move.w      d2,d0
SottFatto2:
        moveq       #16,d1
        bsr.w       dodivu
        and.w       #$00f,d1        ; seleziona solo BLU
        move.w      (a1),d0             ; colore attuale della tab sorgente in d0
        move.w      (a2),d2             ; colore tab destinazione in d2
        and.w       #$00f,d0        ; selez. solo il BLU dal colore tab sorgente
        and.w       #$00f,d2        ; idem per colore tab destinazione
        cmp.w       d0,d2               ; i BLU sorg. e destinazione sono uguali?
        bhi.b       SommaD1             ; se d0>d2, somma d1 a d0
        beq.b       OkBlu               ; se sono uguali, ok
        sub.w       d1,d0               ; d0=d0-d1
        bra.b       OkBlu
SommaD1:
        add.w       d1,d0               ; d0=d0+d1
OkBlu:
        move.w      d0,ColoreFinale     ; Salviamo il BLU finale

; VERDE

        move.w      (a1),d0             ; colore attuale della tab sorgente in d0
        move.w      (a2),d2             ; colore tab destinazione in d2
        and.l       #$0f0,d0        ; selez. solo il VERDE dal colore tab sorgente
        and.l       #$0f0,d2        ; idem per colore tab destinazione
        cmp.w       d2,d0               ; i VERDE sorg. e destinazione sono uguali?
        bhi.b       SottraiD2v      ; se d2>d0, FadCh1a
        beq.b       SottraiD2v      ; se sono uguali, Sottrai d2
        sub.w       d0,d2               ; se d2<d0, subba d0 a d2!
        bra.b       SottFattov
SottraiD2v:
        sub.w       d2,d0               ; altrimenti subba d2 a d0!
        bra.b       SottFatto2v

SottFattov:
        move.w      d2,d0
SottFatto2v:
```

```
        moveq       #16,d1
        bsr.w       dodivu
        and.w       #$0f0,d1        ; seleziona solo VERDE
        move.w       (a1),d0               ; colore attuale della tab sorgente in d0
        move.w       (a2),d2               ; colore tab destinazione in d2
        and.w       #$0f0,d0        ; selez. solo il VERDE dal colore tab sorgente
        and.w       #$0f0,d2        ; idem per colore tab destinazione
        cmp.w       d0,d2               ; i VERDE sorg. e destinazione sono uguali?
        bhi.b       SommaD1v        ; se d0>d2, somma d1 a d0
        beq.b       OkVERDE               ; se sono uguali, ok
        sub.w       d1,d0               ; d0=d0-d1
        bra.b       OkVERDE
SommaD1v:
        add.w       d1,d0               ; d0=d0+d1
OkVERDE:
        or.w        d0,ColoreFinale        ; con l'OR sistema la componente verde

;        ROSSO

        move.w       (a1),d0               ; colore attuale della tab sorgente in d0
        move.w       (a2),d2               ; colore tab destinazione in d2
        and.l       #$f00,d0        ; selez. solo il ROSSO dal colore tab sorgente
        and.l       #$f00,d2        ; idem per colore tab destinazione
        cmp.w       d2,d0               ; i ROSSO sorg. e destinazione sono uguali?
        bhi.b       SottraiD2r        ; se d2>d0, FadCh1a
        beq.b       SottraiD2r        ; se sono uguali, Sottrai d2
        sub.w       d0,d2               ; se d2<d0, subba d0 a d2!
        bra.b       SottFattor
SottraiD2r:
        sub.w       d2,d0               ; altrimenti subba d2 a d0!
        bra.b       SottFatto2r

SottFattor:
        move.w       d2,d0
SottFatto2r:
        moveq       #16,d1
        bsr.w       dodivu
        and.w       #$f00,d1        ; seleziona solo ROSSO
        move.w       (a1),d0               ; colore attuale della tab sorgente in d0
        move.w       (a2),d2               ; colore tab destinazione in d2
        and.w       #$f00,d0        ; selez. solo il ROSSO dal colore tab sorgente
        and.w       #$f00,d2        ; idem per colore tab destinazione
        cmp.w       d0,d2               ; i ROSSO sorg. e destinazione sono uguali?
        bhi.b       SommaD1r        ; se d0>d2, somma d1 a d0
        beq.b       OkROSSO               ; se sono uguali, ok
        sub.w       d1,d0               ; d0=d0-d1
        bra.b       OkROSSO
SommaD1r:
        add.w       d1,d0               ; d0=d0+d1
OkROSSO:
        or.w        d0,ColoreFinale        ; con l'OR sistema la componente rossa

;        Metti il colore in copperlist!

        move.w       ColoreFinale(PC),(a0)        ; e metti il colore finale in copper!

ProssimoColore:
        dbra        d6,f2main        ; ripeti per ogni colore

        addq.w       #1,FaseDelFade        ; sistema per la prossima volta la fase da fare
nocrs:
        rts
```

```
***
*        Input -> D0 = Numeratore
*                 D1 = Denominatore        (16)
*                 D4 = * fattore di moltiplicazione
*
* Output -> D1 = Risultato
***

DoDivu:
        divu.w        d1,d0        ; divisione per 16, non ottimizzabilr con lsr
        move.l        d0,d1
        swap        d1
        move.l        #$31000,d2        ;$10003 (65539) divu 16
        moveq        #0,d3
        move.w        d1,d3
        mulu.w        d3,d2
        move.w        d2,d1

        and.l        #$ffff,d1
        mulu.w        d4,d1                ; moltiplica per la fase del fade
        swap        d1
        mulu.w        d4,d0                ; moltiplica per la fase del fade
        add.w        d0,d1
        and.l        #$ffff,d1
        rts

FaseDelFade:                ; fase attuale del fade (0-16)
        dc.w        0

;        In questa label viene salvato il colore finale ogni volta

ColoreFinale:
        dc.w        0

; ---

Cstart1:
        dcb.w        15,0        ; partiamo dal nero
Cend1:
        dc.w $fff,$200,$310,$410,$620,$841,$a73                ; e arriviamo ai colori
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446
;=----------

Cstart2:
        dc.w $fff,$200,$310,$410,$620,$841,$a73                ; partiamo dai colori
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446
Cend2:
        dcb.w        15,0                        ; e finiamo al nero
;=----------

Cstart3:
        dcb.w        15,$FFF        ; partiamo dal BIANCO
Cend3:
        dc.w $fff,$200,$310,$410,$620,$841,$a73                ; e arriviamo ai colori
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446
;=----------

Cstart4:
        dc.w $fff,$200,$310,$410,$620,$841,$a73                ; partiamo dai colori
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446
Cend4:
```

```
        dc.w $fff,$020,$031,$041,$062,$184,$3a7                ; e arriviamo colori
        dc.w $5b9,$6db,$7dc,$111,$222,$433,$b99,$644        ; diversi! (tono verde)
;=----------

Cstart5:
        dc.w $fff,$020,$031,$041,$062,$184,$3a7                ; partiamo dai colori
        dc.w $5b9,$6db,$7dc,$111,$222,$433,$b99,$644        ; diversi! (tono verde)
Cend5:
        dc.w $fff,$002,$013,$014,$026,$148,$37a                ; ad altri ancora
        dc.w $59b,$6bd,$7cd,$111,$222,$334,$99b,$446        ; diversi! (tono blu)
;=----------


; il $180, color0, e' $000, dunque non cambia! La tabella parte dal color1

TabColoriPic:
        dc.w $fff,$200,$310,$410,$620,$841,$a73
        dc.w $b95,$db6,$dc7,$111,$222,$334,$99b,$446


*****************************************************************************
;                        Copper List
*****************************************************************************
        section        copper,data_c                ; Chip data

Copperlist:
        dc.w        $8E,$2c81        ; DiwStrt - window start
        dc.w        $90,$2cc1        ; DiwStop - window stop
        dc.w        $92,$38                ; DdfStart - data fetch start
        dc.w        $94,$d0                ; DdfStop - data fetch stop
        dc.w        $102,0                ; BplCon1 - scroll register
        dc.w        $104,0                ; BplCon2 - priority register
        dc.w        $108,0                ; Bpl1Mod - modulo pl. dispari
        dc.w        $10a,0                ; Bpl2Mod - modulo pl. pari

                ; 5432109876543210
        dc.w        $100,%0100001000000000        ; BPLCON0 - 4 planes lowres (16 colori)

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane

; i primi 16 colori sono per il LOGO

CopColors:
        dc.w $180,0,$182,0,$184,0,$186,0
        dc.w $188,0,$18a,0,$18c,0,$18e,0
        dc.w $190,0,$192,0,$194,0,$196,0
        dc.w $198,0,$19a,0,$19c,0,$19e,0

;        Mettiamo un poco di sfumature per la scenografia...

        dc.w        $8007,$fffe        ; Wait - $2c+84=$80
        dc.w        $100,$200        ; bplcon0 - no bitplanes
        dc.w        $180,$003        ; color0
        dc.w        $8207,$fffe        ; wait
        dc.w        $180,$005        ; color0
        dc.w        $8507,$fffe        ; wait
```

```
        dc.w        $180,$007       ; color0
        dc.w        $8a07,$fffe       ; wait
        dc.w        $180,$009       ; color0
        dc.w        $9207,$fffe       ; wait
        dc.w        $180,$00b       ; color0

        dc.w        $9e07,$fffe       ; wait
        dc.w        $180,$999       ; color0
        dc.w        $a007,$fffe       ; wait
        dc.w        $180,$666       ; color0
        dc.w        $a207,$fffe       ; wait
        dc.w        $180,$222       ; color0
        dc.w        $a407,$fffe       ; wait
        dc.w        $180,$001       ; color0

        dc.l        $ffff,$fffe       ; Fine della copperlist


*****************************************************************************
;                                DISEGNO
*****************************************************************************

        section        gfxstuff,data_c

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).

Logo1:
        incbin        'logo320*84*16c.raw'

        end
```

Ecco una routine che "trasforma" i colori come vogliamo.
Il principio di funzionamento e' piu' complesso di un normale fade, quindi
basta che capiate come si puo' usare. Se volete friggervi il cervello,
comunque, ho messo dei commenti.

24.7 Lezione8g

```
; Lezione8g.s - Prova di "pavimento" in parallasse - 10 livelli.

*****************************************************************************
*        PARALLAX 0.5        Copyright © 1994 by Federico "GONZO" Stango           *
*                         Modificato da Fabio Ciucci                          *
*****************************************************************************

        SECTION        MAINPROGRAM,CODE        ; Sezione Codice: ovunque in memoria

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA
;                -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
```

```
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
        move.l        #PARALLAXPIC,d0               ; Carico indirizzo Pic in d0
        lea         BPLPointers,a1                ; Indirizzo dei pointers ai planes
        moveq         #5-1,d1                       ; NumPiani-1 per il DBRA
        move.w         #40*56,d2                ; Bit per piano in d2
        bsr.w         PointBpls                ; Chiamiamo la subroutine PointBpls

        MOVE.W         #DMASET,$96(a5)               ; DMACON - abilita bitplane, copper
        move.l         #MyCopList,$80(a5)        ; Puntiamo la nostra COP
        move.w         d0,$88(a5)                ; Facciamo partire la COP
        move.w         #0,$1fc(a5)               ; Disattiva l'AGA
        move.w         #$c00,$106(a5)               ; Disattiva l'AGA
        move.w         #$11,$10c(a5)               ; Disattiva l'AGA

MainLoop:
        MOVE.L         #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L         #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L         4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L         D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L         D2,D0                ; aspetta la linea $130 (304)
        BNE.S         Waity1

        bsr.s         ParallaxFX                ; Salta alla subroutine "Parallax"

        MOVE.L         #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L         #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L         4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L         D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L         D2,D0                ; aspetta la linea $130 (304)
        BEQ.S         Aspetta

        btst        #6,$bfe001                ; LMB premuto?
        bne.s         MainLoop                ; Se "NO" ricomincia
        rts

*****************************************************************************
*                        Parte dedicata alle subroutines                    *
*****************************************************************************

; Questa e' la routine di parallasse. Funziona in un modo molto semplice,
; infatti non fa che modificare i valori dei 10 BPLCON1 ($dff102) posti uno
; sotto l'altro con dei WAIT nella zona del "pavimento". Ebbene, abbiamo gia'
; visto nelle lezioni precedenti come si possa far "ondeggiare" una figura
; utilizzando una copperlist con molti BPLCON1 ($dff102), i quali possono
; spostare verso destra la schermata per un massimo di 15 pixel, col valore
; $FF, mentre con $00 lo spostamento e' nullo. Ora, se anziche' ondulare una
; figura vogliamo fare in modo che sembri che scorra all'infinito, il problema
; che si pone e' quello che al massimo possiamo far scorrere la figura di 15
; pixel, e 15 pixel non sono l'infinito. Potremmo anche fare una figura larga
; un chilometro in memoria, e fare lo scroll usando anche i bplpointers, ma
; non sarebbe economico. Dunque dobbiamo fare uno scroll, che sembri infinito,
; verso destra, con una figura larga solo 320 pixel. Il "trucco" e' questo: se
; la figura in questione e' fatta a "blocchetti" uguali larghi 16 pixel ognuno
; si puo' ingannare l'occhio nascondendogli il fatto che spostiamo di soli 15
; pixel e poi "ripartiamo" da zero. Infatti basta avere una "mattonella" larga
```

```
; un tot di pixel, ad esempio 16, ripetuta per tutto lo schermo, per simulare
; uno scorrimento continuo, basta, appunto, spostare il tutto un pixel alla
; volta verso destra, fino a che l'ultima "mattonella" a destra e' uscita dal
; bordo e ne e' "entrata" una intera dal bordo sinistro: anziche' scattare
; al sedicesimo pixel, impossibile tra l'altro per la limitazione del BPLCON1,
; basta "indietreggiare" di 15 pixel, ripartire da zero, e' la situazione in
; realta' sara' la stessa di quella che si sarebbe verificata spostando di un
; pixel avanti: l'ultima mattonella sulla destra sarebbe sparita completamente
; e la prima a sinistra sarebbe "entrata" completamente. Per fare dei livelli
; che scorrano a velocita' diverse basta fare in modo che ognuno di questi
; livelli vengano spostati il primo ogni 25 fotogrammi, il secondo ogni 16, e
; cosi' via, fino agli ultimi che devono non solo muoversi ogni fotogramma, ma
; scattare 2 o 4 pixel alla volta per andare piu' veloce di 50Hz.
; Per contare da quanti fotogrammi e' stato eseguito lo scroll di ogni livello
; sono state usati dei contatori che vengono incrementati ogni fotogramma, poi
; con un CMP si controlla se si e' aspettato il numero di fotogrammi giusto.
; PxCounter1,2... sono i contatori, Parallax1,2... sono i BPLCON1 in COPLIST


ParallaxFX:
para1:
        addq.b        #$01,PxCounter1        ; Incremento il Contatore di Parallasse 1
        cmpi.b        #25,PxCounter1        ; Contatore velocita'=25?
        bne.s         Para2                ; non ancora 25 fotogrammi...
        clr.b         PxCounter1        ; passati 25 fotogrammi! azzera il contatore
        cmp.b         #$ff,Parallax1        ; Abbiamo raggiunto il valore di scroll
                                ; massimo? (15 pixel verso destra)
        beq.s         riazzera1        ; se si, dobbiamo ripartire da zero!
        add.b         #$11,Parallax1        ; se non ancora, sposta il livello 1
        bra.s         para2
riazzera1:
        clr.b         Parallax1        ; riparti da zero con lo scroll
Para2:
        addq.b        #$01,PxCounter2        ; Incremento il Contatore di Parallasse 2
        cmpi.b        #16,PxCounter2        ; Contatore velocita'=16?
        bne.s         Para3                ; (I commenti sarebbero analoghi a Para1)
        clr.b         PxCounter2
        cmp.b         #$ff,Parallax2
        beq.s         riazzera2
        add.b         #$11,Parallax2        ; sposta il livello di parallasse 2
        bra.s         para3
riazzera2:
        clr.b         Parallax2
Para3:
        addq.b        #$01,PxCounter3        ; Incremento il Contatore di Parallasse 3
        cmpi.b        #10,PxCounter3        ; Contatore velocita'=10?
        bne.s         Para4
        clr.b         PxCounter3
        cmp.b         #$ff,Parallax3
        beq.s         riazzera3
        add.b         #$11,Parallax3        ; sposta il livello di parallasse 3
        bra.s         para4
riazzera3:
        clr.b         Parallax3
Para4:
        addq.b        #$01,PxCounter4        ; Incremento il Contatore di Parallasse 4
        cmpi.b        #5,PxCounter4        ; Contatore velocita'=5?
        bne.s         Para5
        clr.b         PxCounter4
        cmp.b         #$ff,Parallax4
        beq.s         riazzera4
        add.b         #$11,Parallax4        ; sposta il livello di parallasse 4
```

```
        bra.s       para5
riazzera4:
        clr.b       Parallax4
Para5:
        addq.b       #$01,PxCounter5        ; Incremento il Contatore di Parallasse 5
        cmpi.b       #4,PxCounter5        ; Contatore velocita'=4?
        bne.s       Para6
        clr.b       PxCounter5
        cmp.b       #$ff,Parallax5
        beq.s       riazzera5
        add.b       #$11,Parallax5        ; sposta il livello di parallasse 5
        bra.s       para6
riazzera5:
        clr.b       Parallax5
Para6:
        addq.b       #$01,PxCounter6        ; Incremento il Contatore di Parallasse 6
        cmpi.b       #3,PxCounter6        ; Contatore velocita'=3?
        bne.s       Para7
        clr.b       PxCounter6
        cmp.b       #$ff,Parallax6
        beq.s       riazzera6
        add.b       #$11,Parallax6        ; sposta il livello di parallasse 6
        bra.s       para7
riazzera6:
        clr.b       Parallax6
Para7:
        addq.b       #$01,PxCounter7        ; Incremento il Contatore di Parallasse 7
        cmpi.b       #2,PxCounter7        ; Contatore velocita'=2?
        bne.s       Para8
        clr.b       PxCounter7
        cmp.b       #$ff,Parallax7
        beq.s       riazzera7
        add.b       #$11,Parallax7        ; sposta il livello di parallasse 7
        bra.s       Para8
riazzera7:
        clr.b       Parallax7
                    ; DA NOTARE CHE PARA8,PARA9,PARA10 DEVONO
                    ; ESSERE ESEGUITI OGNI FRAME, DUNQUE NON
Para8:                   ; OCCORRE UN CONTATORE DI RITARDO!
        cmp.b       #$ff,Parallax8        ; Abbiamo raggiunto il massimo scroll?
        bne.s       NonRiazzera8
        clr.b       Parallax8        ; azzera parallax8
        bra.s       para9
NonRiazzera8:
        add.b       #$11,Parallax8        ; sposta il livello di parallasse 8
Para9:
        cmp.b       #$ee,Parallax9        ; Abbiamo raggiunto il massimo scroll?
                    ; Il massimo e' $ee e non $ff perche' questo
                    ; livello deve scattare a passi di 2 ogni
                    ; frame, per cui: 00,22,44,66,88,aa,cc,ee
        bne.s       NonRiazzera9
        clr.b       Parallax9        ; azzera parallax9
        bra.s       Para10
NonRiazzera9:
        add.b       #$22,Parallax9        ; sposta il livello di parallasse 9 (2 pixel!)
Para10:
        cmp.b       #$cc,Parallax10        ; Abbiamo raggiunto il massimo scroll?
                    ; Il massimo e' $cc e non $ff perche' questo
                    ; livello deve scattare a passi di 4 ogni
                    ; frame, per cui: 00,44,88,cc
        bne.s       NonRiazzera10
        clr.b       Parallax10        ; azzera parallax10
```

```
        bra.s       ParaFinito
NonRiazzera10:
        add.b       #$44,Parallax10        ; sposta il livello di parallasse 10 (4 pixel)
ParaFinito:
        rts

; Le variabili usate per contare i ritardi per i primi 7 livelli, che devono
; essere spostati una volta ogni 25,16,10 ecc. fotogrammi.

PxCounter1:       dc.b      $00
PxCounter2:       dc.b      $00
PxCounter3:       dc.b      $00
PxCounter4:       dc.b      $00
PxCounter5:       dc.b      $00
PxCounter6:       dc.b      $00
PxCounter7:       dc.b      $00
        even

; SubRoutine per puntare i bitplanes...

************* d0=Indirizzo picture               | d2=Numero di bit per piano
* PointBpls * d1=NumPiani-1 per il DBRA              |
************* a1=Indirizzo puntatori ai planes       |
PointBpls:
        move.w      d0,6(a1)        ; .w bassa nella .w giusta della CopperList
        swap        d0              ; Scambia le 2 .w di d0
        move.w      d0,2(a1)        ; .w alta nella .w giusta della CopperList
        swap        d0              ; Rimetto a posto d0
        add.l       d2,d0               ; Aggiungo lungh. bitplane a d0 - pross. bitp.
        addq.w      #8,a1                ; indirizzo dei prossimi bplpointers
        dbra        d1,PointBpls       ; Ricomincio il ciclo
        rts

*****************************************************************************
        SECTION       PROGDATA,DATA_C               ; Dati: Questa va in CHIPRAM           *
*****************************************************************************

MyCopList:
        dc.w        $8e,$2c91       ; DiwStrt (finestra video fatta
                                    ; partire 16 pixel piu' a destra per
                                    ; coprire l'orrore (ehm. errore)
        dc.w        $90,$2cc1       ; DiwStop
        dc.w        $92,$0038       ; DdfStart
        dc.w        $94,$00d0       ; DdfStop
        dc.w        $102,0              ; BplCon1
        dc.w        $104,0              ; BplCon2
        dc.w        $108,0              ; BplMod1
        dc.w        $10a,0              ; BplMod2
        dc.w        $100,$200       ; No Bitplanes...

Rainbow:
        dc.w        $180,$a9c
        dc.w        $eb07,$fffe
        dc.w        $180,$bad
        dc.w        $ed07,$fffe
        dc.w        $180,$cbe
        dc.w        $ef07,$fffe
        dc.w        $180,$dce
        dc.w        $f107,$fffe
        dc.w        $180,$ede
        dc.w        $f307,$fffe
        dc.w        $180,$fef
```

```
        dc.w        $f407,$fffe        ; wait - aspetta

        dc.w        $100,%0101001000000000        ; LowRes 32Colors

BPLPointers:
        dc.w        $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w        $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w        $e8,$0000,$ea,$0000        ;terzo         bitplane
        dc.w        $ec,$0000,$ee,$0000        ;quarto         bitplane
        dc.w        $f0,$0000,$f2,$0000        ;quinto         bitplane

        dc.w        $0180
Colours:
        dc.w        $fff,$182,$f10,$184,$f21,$186,$f42
        dc.w        $188,$f53,$18a,$f63,$18c,$f74,$18e,$f85
        dc.w        $190,$f96,$192,$fa6,$194,$fb7,$196,$fb8
        dc.w        $198,$fc9,$19a,$f21,$19c,$f10,$19e,$f00
        dc.w        $1a0,$eff,$1a2,$eff,$1a4,$dff,$1a6,$dff
        dc.w        $1a8,$cff,$1aa,$bef,$1ac,$bef,$1ae,$adf
        dc.w        $1b0,$9df,$1b2,$9cf,$1b4,$8bf,$1b6,$7bf
        dc.w        $1b8,$7af,$1ba,$69f,$1bc,$68f,$1be,$57f


; ECCO LA PARTE DI COPPERLIST RESPONSABILE DELLA PARALLASSE:

        dc.w        $f507,$fffe        ; Wait linea $f5
        dc.w        $180,$f52        ; Color0 - sfondo arancione per "mimetizzarsi"
                            ; con la figura

        dc.w        $102                ; BPLCON1
        dc.b        $00                ; byte alto, non usato
Parallax1:
        dc.b        $00                ; byte basso, valore di scroll!!!!

        dc.w        $f607,$fffe        ; wait
        dc.w        $102                ; BPLCON1
        dc.b        $00                ; eccetera, per ogni "livello"
Parallax2:
        dc.b        $00

        dc.w        $f807,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax3:
        dc.b        $00

        dc.w        $fb07,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax4:
        dc.b        $00

        dc.w        $ff07,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax5:
        dc.b        $00

        dc.w        $ffdf,$fffe        ; per superare la linea $FF

        dc.w        $0407,$fffe
```

```
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax6:
        dc.b        $00

        dc.w        $0b07,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax7:
        dc.b        $00

        dc.w        $1207,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax8:
        dc.b        $00

        dc.w        $1a07,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax9:
        dc.b        $00

        dc.w        $2307,$fffe
        dc.w        $102        ; BPLCON1
        dc.b        $00
Parallax10:
        dc.b        $00

        dc.w        $2c07,$fffe
        dc.w        $180,$f30

        dc.w        $FFFF,$FFFE        ; Fine CopList

; L'immagine e' larga 320 pixel e alta 56, a 5 bitplanes (32 colori)

PARALLAXPIC:
        incbin        "Lava320*56*5.Raw"        ; Includo l'immagine.

        END
```

Questo listatuccio lo ha fatto il mio allievo "Gonzo" una volta letta la LEZIONE5. Mi telefono' chiedendomi come fare un parallasse, e gli ho risposto prontamente che una volta letta la lezione 5 sarebbe stato in grado di farne uno, nonostante non ci fosse un listato specifico. Come vedete ha intuito bene come fare. C'e' pero' un erroruccio, facilmente removibile, ossia il fatto che avviene il classico "errore" dello scroll nei primi 16 pixel a sinistra. La figura infatti e' larga solo 320 pixel, per cui quando sposta i vari livelli di parallasse porta via anche la parte sinistra del livello in questione. Per vedere l'errore riportate a livelli normali il DiwStart, che in questo listato e' modificato per "tappare" il problema:

```
        dc.w        $8e,$2c91        ; DiwStrt (finestra video fatta
                                     ; partire 16 pixel piu' a destra per
                                     ; coprire l'orrore ehm. errore)
```

Sostituitelo con lo standard $2c81 e noterete il danno sulla sinistra.
Per ovviare definitivamente al problema, basta fare come abbiamo fatto per lo scroll di una figura piu' grande dello schermo: occore ridisegnare la figura della pavimentazione facendola 16 pixel piu' larga, ossia 336 pixel, cioe' dobbiamo aggiungere una "mattonella" in piu'. A questo punto occorre puntare la figura ricordandosi di questo "allargamento", agendo proprio come nel caso

```
dello scroll "gigante", lasciando agire l'errore nei 16 pixel "fuori schermo"
sulla sinistra.
Questa e' solo una base per un pavimento in parallasse. Si potrebbe anche fare
uno scorrimento piu' fluido, linea per linea, calcolandolo con precisione
matematica con una tabella, e si potrebbe cambiare anche la palette per ogni
livello per sfumare maggiormente i colori. Se avete voglia poi di aggiungere
le nuvole in parallasse, le montagnine e gli uccellini, vi prego di spedirmi
l'opera!
```

24.8 Lezione8h

```
; Lezione8h.s - Un Utilizzo della routine UniMuoviSprite per fare un pannello
;               di controllo con gadgets

        SECTION        CiriCop,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110100000        ; solo copper,bitplane,sprite DMA
;                -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
        bsr.w        PuntaFig1        ; Puntiamo la Fig.1
        bsr.w        PuntaFigBase        ; Puntiamo la Fig.base

        move.l        #BufferVuoto,d0        ; punta uno spazio azzerato dove sara'
        LEA        BPLPOINTER2,A1        ; stampato il testo
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        move.b        $dff00a,mouse_y                ; Diamo alle variabili mouse_y-x il
        move.b        $dff00b,mouse_x                ; valore attuale in lettura del mouse

;*****************************************************************************
;                                LOOP PRINCIPALE
;*****************************************************************************

Clear:
        clr.b        Azione                ; Riazzera le variabili
```

```
        clr.b       TastoAzionato
        clr.b       EsciVar

Programma:
        ****1
        btst       #6,$bfe001       ; Tasto Sinistro del mouse premuto? Se no
        bne.s       Contprog        ; continua il programma, altrimenti:
        bsr.w       CheckAzione       ; Controlla quale tasto abbiamo premuto
        cmpi.b       #1,TastoAzionato; Se abbiamo premuto uno dei "tasti" la
        beq.s       Comando               ; variante "TastoAzionato" e'=1; andiamo
                            ; a controllare quale tasto abbiamo cliccato!
Contprog:
        bsr.w       MuoviFreccia        ; routine che legge/muove il mouse
        bra.s       Programma        ; Fine del programma: ritorniamo da capo!

;******************************************************************************
;        Routine "Comando" di interpretazione del tasto premuto
;******************************************************************************

; Nella variabile "AZIONE" troviamo un valore, il quale è stato immesso
; precedentemente dalla routine "CheckAzione". Controllando il suo valore
; possiamo sapere quale tasto abbiamo "clicckato", ed andare ad eseguire il
; suo corrispettivo programmino.

Comando:
        cmpi.b       #$f,Azione        ; Se azione e' "f", abbiamo cliccato sul tasto
        beq.s       Verde               ; VERDE
        cmpi.b       #$e,Azione        ; Se azione e' "e", abbiamo cliccato sul tasto
        beq.w       Rosso               ; VERDE
        cmpi.b       #$d,Azione        ; Se azione e' "d", abbiamo cliccato sul tasto
        beq.w       Giallo               ; GIALLO
        cmpi.b       #7,Azione        ; Se azione e' "7", abbiamo cliccato sul tasto
        beq.w       Music_On        ; Music_On
        cmpi.b       #6,Azione        ; Se azione e' "6", abbiamo cliccato sul tasto
        beq.w       Music_Off        ; Music_Off
        cmpi.b       #5,Azione        ; Se azione e' "5", abbiamo cliccato sul tasto
        beq.w       Esci               ; Quit
        cmpi.b       #4,Azione        ; Se azione e' "4", abbiamo cliccato sul tasto
        beq.w       PalNtsc               ; PalNtsc
        cmpi.b       #3,Azione        ; Se azione e' "3", abbiamo cliccato sul tasto
        beq.w       Piu                ; Piu
        cmpi.b       #$2,Azione        ; Se azione e' "2", abbiamo cliccato sul tasto
        beq.w       Meno                ; Meno
;        cmpi.b       #1,Azione        ; Se azione e' "1", abbiamo cliccato sul tasto
        bra.w       GiuSu               ; GiuSu (In verita', e' rimasto solo questa
                            ; possibilita',dunque ci saltiamo direttamente

;******************************************************************************

Verde:
        bsr.w       MuoviFreccia        ; routine che legge/muove il mouse
        lea        Barra+6,a6        ; Per far tornare la moltiplicazione
        move.b       #$1,ColorB        ; Memoriziamo quale colore stiamo visualizzando
        move.w       #$0030,(a6)        ; Cambiamo i COLORI della barra (la distanza
        move.w       #$0060,8(a6)        ; tra un wait e l'altro e' di 8 bytes)
        move.w       #$0090,8*2(a6)
        move.w       #$00c0,8*3(a6)
        move.w       #$00f0,8*4(a6)
        move.w       #$00c0,8*5(a6)
        move.w       #$0090,8*6(a6)
        move.w       #$0060,8*7(a6)
        move.w       #$0030,8*8(a6)
```

```
        bra.w        Clear               ; Ritorniamo da capo!

;***************************************************************************

Rosso:
        bsr.w        MuoviFreccia        ; routine che legge/muove il mouse
        lea        Barra+6,a6        ; Per far tornare la moltiplicazione
        move.b        #$2,ColorB        ; Memoriziamo quale colore stiamo visualizzando
        move.w        #$0300,(a6)        ; Cambiamo i wait della barra (la distanza
        move.w        #$0600,8(a6)        ; tra un wait e l'altro e' di 8 bytes)
        move.w        #$0900,8*2(a6)
        move.w        #$0c00,8*3(a6)
        move.w        #$0f00,8*4(a6)
        move.w        #$0c00,8*5(a6)
        move.w        #$0900,8*6(a6)
        move.w        #$0600,8*7(a6)
        move.w        #$0300,8*8(a6)
        bra.w        Clear               ; Ritorniamo da capo!

;***************************************************************************

Giallo:
        bsr.w        MuoviFreccia        ; routine che legge/muove il mouse
        lea        Barra+6,a6        ; Per far tornare la moltiplicazione
        clr.b        ColorB               ; Memoriziamo quale colore stiamo visualizzando
        move.w        #$0310,(a6)        ; Cambiamo i wait della barra (la distanza
        move.w        #$0640,8(a6)        ; tra un wait e l'altro e' di 8 bytes)
        move.w        #$0970,8*2(a6)
        move.w        #$0ca0,8*3(a6)
        move.w        #$0fd0,8*4(a6)
        move.w        #$0ca0,8*5(a6)
        move.w        #$0970,8*6(a6)
        move.w        #$0640,8*7(a6)
        move.w        #$0310,8*8(a6)
        bra.w        Clear               ; Ritorniamo da capo!

;***************************************************************************

PaNtFlag:
        dc.w        0

PalNtsc:
        bchg.b        #1,PaNtflag
        btst.b        #1,PaNtflag
        beq.s        VaiPal
        move.w        #0,$1dc(a5)        ; BEAMCON0 (ECS+) Risuluzione video NTSC
        bra.w        Clear               ; Ritorniamo da capo!
VaiPal
        move.w        #$20,$1dc(a5)        ; BEAMCON0 (ECS+) Risoluzione video PAL
        bra.w        Clear


;***************************************************************************

; Ricordiamoci di mettere SEMPRE la routine "MuoviFreccia" nei punti, ad
; esempio come il seguente, che non ritorna ad eseguire il programma principale
; fino a che non è premuto il tasto sinistro del mouse. Se fosse ommesso,
; il mouse non si muoverebbe fino a che non rilasciamo il tasto del mouse!

Piu:
        bsr.w        MuoviFreccia        ; routine che legge/muove il mouse
        lea        barra,a6        ; Mettiamo in a5, l'indirizzo di "BARRA", così
```

```
                            ; evitando di riscriverla ogni volta, ed
                            ; inoltre risulta più veloce l'esecuzione!
        cmpi.b       #$84,8*9(a6)        ; Siamo arrivati alla linea $84?
        beq.s        FinePiu               ; Se si, siamo in cima, e fermiamoci.
        addq.b       #1,(a6)                ; Muoviamo la posizione della barra di un pixel
        addq.b       #1,8(a6)        ; alla volta
        addq.b       #1,8*2(a6)
        addq.b       #1,8*3(a6)
        addq.b       #1,8*4(a6)
        addq.b       #1,8*5(a6)
        addq.b       #1,8*6(a6)
        addq.b       #1,8*7(a6)
        addq.b       #1,8*8(a6)
        addq.b       #1,8*9(a6)

**2
        btst.b       #6,$bfe001        ; Finchè il tasto sinistro non è rilasciato
        beq.s        Piu                ; la barra continua a muoversi, nonstante il
                            ; mouse non sia più sopra il tasto "+":
                            ; Provate ad aggiundere sotto la prima linea
                            ; "bsr.w Muovifreccia" la label "PIU2", e
                            ; cambiare anche la linea sotto ***2 in
                            ; "beq.s Piu2". Nonostante che muoviate il
                            ; mouse, la freccia non si muoverà!

        bra.w        Clear                ; Torniamo da capo!!

; siamo arrivati in fondo? Allora barra BLU

FinePiu:
        bsr.w        MuoviFreccia        ; routine che legge/muove il mouse
        lea        Barra+6,a6        ; Barra in coplist
        move.w       #$0003,(a6)        ; Cambiamo i COLORI della barra in BLU
        move.w       #$0006,8(a6)
        move.w       #$0009,8*2(a6)
        move.w       #$000c,8*3(a6)
        move.w       #$000f,8*4(a6)
        move.w       #$000c,8*5(a6)
        move.w       #$0009,8*6(a6)
        move.w       #$0006,8*7(a6)
        move.w       #$0003,8*8(a6)
        btst.b       #6,$bfe001        ; Finchè il tasto sinistro non è rilasciato
        beq.s        FinePiu               ; la barra continua a muoversi, nonstante il
                            ; mouse non sia più sopra il tasto "+"
        cmp.b        #1,ColorB        ; Controlliamo di quale colore era prima
                            ; la barra tramite la variabile ColorB:
                            ; Se risulta di valore "1", allora la barra è
                            ; di colore verde:
        beq.w        Verde                ; Andiamo alla label VERDE, riporatando così la
                            ; barra al suo colore originario
        cmp.b        #2,ColorB        ; Anche qui, se la variabile risulta di valore
        beq.w        Rosso                ; "2", andiamo alla label ROSSO
        bra.w        Giallo                ; Se non si è verificata nessuna condizione
                            ; precedente, INEVITABILMENTE la barra è
                            ; GIALLA, perchè i colori possibili appunto
                            ; sono  tre: rosso, verde o giallo!

;*****************************************************************************

Meno:
        bsr.w        Muovifreccia        ; Vale il solito discorso sopra, solo che
        lea        barra,a6        ; aggiungere il valore "1" a "barra",
```

```
        cmpi.b      #$36,8*9(a6)        ; Siamo arrivati in fondo?
        beq.s       FineMeno        ; se si ferma tutto e colora la barra di BLU
        subq.b      #1,(a6)                 ; sottraiamo, facendola muovere in direzione
        subq.b      #1,8(a6)        ; opposta (verso l'alto)
        subq.b      #1,8*2(a6)
        subq.b      #1,8*3(a6)
        subq.b      #1,8*4(a6)
        subq.b      #1,8*5(a6)
        subq.b      #1,8*6(a6)
        subq.b      #1,8*7(a6)
        subq.b      #1,8*8(a6)
        subq.b      #1,8*9(a6)
        **3
        btst.b      #6,$bfe001
        beq.s       Meno
        bra.w       Clear               ; Torniamo da capo!!

; siamo arrivati in cima? Allora barra blu!

FineMeno:
        bsr.w       MuoviFreccia        ; routine che legge/muove il mouse
        lea       Barra+6,a6
        move.w      #$0003,(a6)         ; colora di BLU
        move.w      #$0006,8(a6)
        move.w      #$0009,8*2(a6)
        move.w      #$000c,8*3(a6)
        move.w      #$000f,8*4(a6)
        move.w      #$000c,8*5(a6)
        move.w      #$0009,8*6(a6)
        move.w      #$0006,8*7(a6)
        move.w      #$0003,8*8(a6)
        btst.b      #6,$bfe001
        beq.s       FineMeno
        cmpi.b      #$1,ColorB        ; Controlla di quale colore era la barra
        beq.w       Verde
        cmpi.b      #$2,ColorB
        beq.w       Rosso
        bra.w       Giallo

;*****************************************************************************

Music_On:
        move.b      #1,MusicFlag        ; Dando il valore "1" alla variabile MusicFlag,
                            ; ogni qualvolta che la testeremo, sapremo
                            ; quando la musica è stata attivata.
        move.l      a5,-(SP)        ; salva a5 nello stack
        bsr.w       mt_init               ; Saltiamo alla routine che suona la musica
        move.l      (SP)+,a5        ; riprendi a5 dallo stack

        **4
;        bsr.w      MuoviFreccia        ; routine che legge/muove il mouse
        bra.w       Clear               ; Ritorniamo da capo!

;*****************************************************************************

Music_Off:
        clr.b       MusicFlag       ; Dando il valore "0" alla variabile MusicFlag,
                            ; ogni qualvolta che la testeremo, sapremo
                            ; quando la musica è stata disattivata.
        move.l      a5,-(SP)        ; salva a5 nello stack
        bsr.w       mt_end              ; Saltiamo alla routine che ferma la musica
        move.l      (SP)+,a5        ; riprendi a5 dallo stack
```

```
        **5
;        bsr.w        MuoviFreccia        ; routine che legge/muove il mouse
        bra.w        Clear               ; Ritorniamo da capo!

;******************************************************************************
;                        Rirtono alla OldCop
;******************************************************************************

Esci:                                ; Usciamo dal programma!!!
        move.l        a5,-(SP)        ; salva a5 nello stack
        bsr.w        mt_end               ; Spegniamo la musica!!!: Se abbiamo premuto
                                ; il tasto "ESCI", mentre la musica stava
                                ; suonando, succede del casino
        move.l        (SP)+,a5        ; riprendi a5 dallo stack
        rts


*******************************************************************************
*                                Vari BSR                                     *
*******************************************************************************

PuntaFig1:
        MOVE.L        #picture1,d0
        moveq        #4-1,d1                ; 4 bitplane!
        LEA        BPLPOINTERS,A1
POINTBPa:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*84,d0        ; La figura e' alta 84 linee, non 256!!
        addq.w        #8,a1
        dbra        d1,POINTBPa

;        Puntiamo tutti gli sprite allo sprite nullo, per essere sicuri che
;        non ci siano problemi.

        MOVE.L        #SpriteNullo,d0                ; indirizzo dello sprite in d0
        LEA        SpritePointers,a1        ; Puntatori in copperlist
        MOVEQ        #8-1,d1                        ; tutti gli 8 sprite
NulLoop:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        addq.w        #8,a1
        dbra        d1,NulLoop

; Puntiamo il primo sprite

        MOVE.L        #MIOSPRITE0,d0
        LEA        SpritePointers,a1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        rts                        ; Ritorno al BSR

PuntaFigBase:
        MOVE.L        #picturebase,d0
        LEA        BPLPOINTERSbase,A1
        moveq        #0,d1                ; 1 bitplane!
POINTBPbasenew:
```

```
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*105,d0        ; La figura e' alta 105 linee, non 256!!
        addq.w        #8,a1
        dbra        d1,POINTBPbasenew
        rts                        ; Ritorno

*****************************************************************************
; Questa routine controlla se abbiamo premuto in un "bottone"/"gadget" o
; dove non ci sono comandi. Nel caso si sia premuto un "bottone", assegna
; il valore corrispondente al "bottone" premuto alla variabile Azione.
*****************************************************************************


;                 _,'|             _.-'''`-...___..--';)
;                /_ \'.      __..-' ,      ,--...--'''
;               ¶     .`--'''       `     /'
;                `-';'                ;   ; ;
;          __...--''     ___...--_..'  .;.'
;         (,__....----'''       (,..--''
;|||||||||///||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

CheckAzione:

; Prima controllo le posizioni Y

        move.b        #$1,TastoAzionato        ; Ipotiziamo anticipatamente che
                                        ; abbiamo premuto dentro uno dei tasti
                                        ; dando alla variabile "TastoAzionato"
                                        ; il valore "1"
        cmpi.w        #$00fc,Sprite_y                ; La freccia e' sotto la posizione dei
                                        ; tasti?, cioe'
                                        ; Sprite_y e' > di 00fc, Se si:
        bhi.s        rtnCheck                ; Siamo fuori!
        cmpi.w        #$00f1,Sprite_y                ; La freccia e' allineata alla linea
                                        ; del "Cambia colore VERDE"?
;
        bhi.w        Effetto_Verde                ; Se si: vai ad Azione verde
;
        cmpi.w        #$00fe,Sprite_y                ; Siamo tra 00f1 e 00fe? Se si:
        bhi.s        rtnCheck                ; Siamo fuori!
        cmpi.w        #$00e4,Sprite_y                ; La freccia e' allineata alla linea
                                        ; del "Cambia colore ROSSO"?
;
        bhi.w        Effetto_Rosso                ; Se si: vai ad Azione rosso
;
        cmpi.w        #$00e1,Sprite_y                ; Siamo tra 00e1 e 00d7? Se si:
        bhi.s        rtnCheck                ; Siamo fuori!
        cmpi.w        #$00d7,Sprite_y                ; La freccia e' allineata alla linea
                                        ; del "Cambia colore GIALLO"?
;
        bhi.w        Effetto_Giallo                ; Se si: vai ad Azione giallo
;
        cmpi.w        #$00d0,Sprite_y                ; Siamo tra 00d7 e 00d0? Se si:
        bhi.s        rtnCheck                ; Siamo fuori!

        cmpi.w        #$00b0,Sprite_y                ; La freccia e' tra i tasti "+","-",
;                                        ; "Pal-Ntsc","Esci"...
        bhi.s        Azione_Tasti                ; Se si: vai ad Azione_Tasti
;
```

```
rtnCheck:
        clr.b       TastoAzionato               ; Non avendo premuto nessun tasto,
        rts                                  ; evitiamo al programma di perdere
                                          ; tempo facendo subito rileggere
                                          ; la posizione del mouse, tramite la
                                          ; variabile "TastoAzionato"

;*****************************************************************************
; Ora che sappiamo che la Y e' quella di qualche "bottone", controlliamo
; anche se la X e' quella giusta!
;*****************************************************************************

Azione_Tasti:
        cmpi.w       #$0111,Sprite_x               ; La freccia e' oltre il tasto
                                          ; "Musica Off"? Se si:
        bhi.s       rtnCheck                ; Siamo fuori!
        cmpi.w       #$00ea,Sprite_x               ; La freccia e' tra il tasto
                                          ; "Musica Off"? Se si:
        bhi.w       Effetto_Off_Music        ; Vai a Effetto_Off_Music

        cmpi.w       #$00dc,Sprite_x               ; La freccia e' tra 00ea e 00dc? Se si:
        bhi.s       rtnCheck                ; Siamo fuori!
        cmpi.w       #$00b3,Sprite_x               ; La freccia e' sopra il tasto
                                          ; "Musica_On"? Se si:
        bhi.w       Effetto_On_Music        ; Vai a Effetto_On_Music
        cmpi.w       #$00ab,Sprite_x               ; La freccia e' oltre i  tasti
                                          ; "Pal/Ntsc" e "Quit"
        bhi.s       rtnCheck                ; Siamo fuori!
        cmpi.w       #$0077,Sprite_x               ; La freccia e' tra i  tasti
                                          ; "Pal/Ntsc,Quit"?
        bhi.s       Quale_Due2               ; Vediamo quale dei "Pal/Ntsc" o "Quit"
        cmpi.w       #$006c,Sprite_x               ; La freccia e' tra 77 e 6c?
        bhi.s       rtnCheck                ; Siamo fuori!
        cmpi.w       #$005d,Sprite_x               ; La freccia e' tra i tasti
                                          ; "+" e "-"?
        bhi.s       Quale_Due1               ; Vediamo quale dei "+" o "-"
        cmpi.w       #$004f,Sprite_x               ; La freccia e' tra 77 e 6c?
        bhi.s       rtnCheck                ; Siamo fuori!
        cmpi.w       #$003e,Sprite_x               ; La freccia e' sul tasto merda:-><-!!
        bhi.s       Effetto_GiuSu               ; Andiamo ad Effetto_GiuSu
        bra.s       rtnCheck                ; Se non si è verificata nessuna azione
                                          ; allora andiamo a rtnCheck

Quale_Due2:
        cmpi.w       #$00c3,Sprite_y               ; La freccia si trova sopra il tasto
        bhi.w       Effetto_Quit               ; "Quit"? Se si vai a Effetto_Quit
        cmpi.w       #$00bc,Sprite_y               ; La freccia e' tra 00c3 e 00bc?
        bhi.w       rtnCheck                ; Siamo in mezzo ai due tasti!
        cmpi.w       #$00b0,Sprite_y               ; La freccia e' tra 00bc e 00b0?
        bhi.w       Effetto_Pal                ; Andiamo a Effetto_Pal

Quale_Due1:
        cmpi.w       #$00c3,Sprite_y               ; La freccia si trova sopra il tasto
        bhi.s       Effetto_Piu                ; "Piu"? Se si, vai a Effetto_Piu
        cmpi.w       #$00bc,Sprite_y         ; La freccia e' tra 00bc e 00c3?
        bhi.w       rtnCheck                ; Siamo in mezzo ai due tasti!
        cmpi.w       #$00b0,Sprite_y               ; La freccia e' tra 00b0 e 00bc?
        bhi.w       Effetto_Meno               ; Andiamo a Effetto_Meno


;*****************************************************************************
; Ora diamo alla Variabile Azione il valore del tasto che e' premuto
```

```
;*****************************************************************************

Effetto_Verde:                              ; Se si e' verificata questa condizione
        move.b        #$d,Azione               ; vuol dire che ci troviamo sopra la
        rts                                 ; barra VERDE! Allora informiamo il
                                          ; programma che e' stato premuto
                                          ; questo "tasto" tramite la variabile
                                          ; Azione, dandole il valore "d".
Effetto_Rosso:
        move.b        #$e,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Rosso- diamo il valore "e".

Effetto_Giallo:
        move.b        #$f,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Giallo- diamo il valore  "f".

Effetto_GiuSu:
        move.b        #$1,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -GiuSu- diamo il valore di "1".

Effetto_Piu:
        move.b        #$2,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Piu- diamo il valore di "2".

Effetto_Meno:
        move.b        #$3,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Meno- diamo il valore di "3".

Effetto_Pal:
        move.b        #$4,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Pal- diamo il valore di "4".

Effetto_Quit:
        move.b        #$5,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Quit- diamo il valore di "5".

Effetto_Off_Music
        move.b        #$6,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -Off_Misic- diamo il valore di "6".

Effetto_On_Music
        move.b        #$7,Azione               ; Uguale a sopra, solo che per il tasto
        rts                                 ; -On_Music- diamo il valore di "7".

*****************************************************************************
* Routine che legge la posizione del mouse                                  *
* immettendo le coordinate in Mouse_x/Mouse_y - Sprite_x/Sprite_Y           *
*****************************************************************************

LeggiMouse:
        move.b        $a(a5),d1         ; $dff00a - JOY0DAT byte alto
        move.b        d1,d0
        sub.b        mouse_y(PC),d0
        beq.s        no_vert
        ext.w        d0
        add.w        d0,sprite_y
no_vert:
          move.b        d1,mouse_y
        move.b        $b(a5),d1         ; $dff00a - JOY0DAT - byte basso
        move.b        d1,d0
        sub.b        mouse_x(PC),d0
        beq.s        no_oriz
```

```
        ext.w       d0
        add.w       d0,sprite_x
no_oriz:
        move.b       d1,mouse_x
        cmpi.w       #$0021,sprite_x              ; Posizione x minima? (bordo sinistro)
        bpl.b       s1                      ; se non ancora, non occorre bloccare.
        move.w       #$0021,sprite_x              ; Altrimenti, blocchiamolo alla
                                        ; posizione $21.. NON OLTRE!!
s1:
        cmpi.w       #$0004,sprite_y              ; Posizione y minima? (inizio schermo)
        bpl.b       s2                      ; se non ancora, non bloccare
        move.w       #$0004,sprite_y              ; altrimenti inchioda lo sprite al
                                        ; bordo superiore sinistro
s2:
        cmpi.w       #$011d,sprite_x              ; Posizione x massima? (bordo destro)
        ble.b       s3                      ; se non ancora, non occorre bloccare
        move.w       #$011d,sprite_x              ; Altrimenti blocchiamolo a $11d
s3:
        cmpi.w       #$00ff,sprite_y              ; Posizione y massuma? (fondo schermo)
        ble.b       s4                      ; se non ancora, non bloccare
        move.w       #$00ff,sprite_y              ; Altrinenti bloccare a $ff
s4:
        rts

*******************************************************
*               Routine che muove lo sprite0              *
*******************************************************
;        a1 = Indirizzo dello sprite
;        d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;        d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;        d2 = altezza dello sprite

UniMuoviSprite:
        ADD.W       #$2c,d0
        MOVE.b       d0,(a1)
        btst.l       #8,d0
        beq.s       NonVSTARTSET
        bset.b       #2,3(a1)
        bra.s       ToVSTOP
NonVSTARTSET:
        bclr.b       #2,3(a1)
ToVSTOP:
        ADD.w       D2,D0
        move.b       d0,2(a1)
        btst.l       #8,d0
        beq.s       NonVSTOPSET
        bset.b       #1,3(a1)
        bra.b       VstopFIN
NonVSTOPSET:
        bclr.b       #1,3(a1)
VstopFIN:
        add.w       #128,D1
        btst       #0,D1
        beq.s       BitBassoZERO
        bset       #0,3(a1)
        bra.s       PlaceCoords
BitBassoZERO:
        bclr       #0,3(a1)
PlaceCoords:
        lsr.w       #1,D1
        move.b       D1,1(a1)
        rts
```

```
*****************************************************************************
*               LOOP di temporizzazione e aggiornamento sprite                *
*****************************************************************************

MuoviFreccia:

; Questa è una routine di temporizzazione perchè viene usato come riferimento
; il pennello elettronico, a 50Hz (a meno che non si stia usando il sistema
; NTSC! (60Hz!) ). Appunto, il pennello in tutti i computer ha la stessa
; velocità, sia nei vecchi A500 che negli A4000.

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$0fe00,d2        ; linea da aspettare = $fe, ossia 254
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $fe (254)
        BNE.S        Waity1


        tst.b        MusicFlag        ; Se MusicFlag è "0", allora la musica non è
        beq.w        NoMusic2                ; stata accesa, per cui saltiamo la linea
                                ; seguente
        move.l        a5,-(SP)        ; salva a5 nello stack
        bsr.w        mt_music        ; Suona la musica se è stato premuto il tasto
                                ; "On_Music"
        move.l        (SP)+,a5        ; riprendi a5 dallo stack

NoMusic2:
        bsr.w        LeggiMouse        ; Salta alla routine che legge la posizione del
                                ; mouse
        move.w        sprite_y(pc),d0        ; Prepara le coordinate di y
        move.w        sprite_x(pc),d1        ; Prepara le coordinate di x
        lea        miosprite0,a1        ; seleziona lo sprite da muovere
        moveq        #13,d2                ; Prepara la lunghezza dello sprite
        bsr.w        UniMuoviSprite        ; Saltiamo alla routine che muove lo sprite
        bsr.w        PrintCarattere        ; Scrivi il testo

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$0fe00,d2        ; linea da aspettare = $0fe, ossia 254
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $0fe (254)
        BEQ.S        Aspetta

        rts

;*****************************************************************************
; Effetto "speciale" della chiusura e apertura col DIWSTART/STOP
;*****************************************************************************


GiuSu:
        bsr.w        SchermoChiudi        ; Saltiamo alla routine che chiude lo schermo
        bsr.w        SchermoApri        ; Saltiamo alla routine che apre lo schermo
        bra.w        Clear                ; Ritorniamo da capo!

;*****************************************************************************

SchermoChiudi:
```

```
        bsr.w       MuoviFreccia        ; aspetta che sia passato 1 ciclo FRAME!!
        ADDQ.B       #1,DiwYStart        ; Caliamo di un pixel lo schermo superiore
        SUBQ.B       #1,DIWySTOP        ; Aumentiamo di un pixel lo schermo inferiore
        CMPI.b       #$ad,DiwYStart        ; Se abbiamo raggiunto la posizione desiderata,
        beq.s       Finito3               ; allora usciamo, altrimenti azzeriamo la
        bra.s       SchermoChiudi        ; pixel
Finito3:
        rts

SchermoApri:
        bsr.w       MuoviFreccia        ; invece si farlo aumentare, lo facciamo
                            ; diminuire, cioe' invertiamo:
                            ; addq #1,DiwyStart
        SUBQ.B       #5,DiwYStart        ; subq #1,DiwyStop
        ADDQ.B       #5,DIWySTOP        ; rispettivamente con
        CMPI.B       #$2c,DiwYStop        ; subq #5,DiwyStart
        beq.w       Finito4               ; addq #5,DiwyStop
        bra.s       SchermoApri
Finito4:
        rts


*****************************************************************************
*                                 Dati                                      *
*****************************************************************************
Azione:
        dc.l        0
TastoAzionato:
        dc.l        0
EsciVar
        dc.l        0
ColorB:
        dc.b        2
        even

MusicFlag:
        dc.w        0

SPRITE_Y:        dc.w        $a0        ; qui viene memorizzata la Y dello sprite
                            ; Cambiando questo valore possiamo cambiare Y
                            ; la posizione iniziale del muose
SPRITE_X:        dc.w        0        ; qui viene memorizzata la X dello sprite
                            ; Cambiando questo valore possiamo cambiare X
                            ; la posizione iniziale del muose
MOUSE_Y:        dc.b        0        ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b        0        ; qui viene memorizzata la X del mouse

*****************************************************************************
;                        Routine di Print
*****************************************************************************

PRINTcarattere:
        MOVE.L       PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ       #0,D2                ; Pulisci d2
        MOVE.B       (A0)+,D2        ; Prossimo carattere in d2
        CMP.B       #$ff,d2                ; Segnale di fine testo? ($FF)
        beq.s       FineTesto        ; Se si, esci senza stampare
        TST.B       d2                ; Segnale di fine riga? ($00)
        bne.s       NonFineRiga        ; Se no, non andare a capo

        ADD.L       #40*7,PuntaBITPLANE        ; ANDIAMO A CAPO
        ADDQ.L       #1,PuntaTesto                ; primo carattere riga dopo
```

```
                                    ; (saltiamo lo ZERO)
        move.b      (a0)+,d2            ; primo carattere della riga dopo
                                    ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B       #$20,D2             ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        LSL.W       #3,D2               ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L      D2,A2
        ADD.L       #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L      PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B      (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
        MOVE.B      (A2)+,40(A3)        ; stampa LA LINEA 2  " "
        MOVE.B      (A2)+,40*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B      (A2)+,40*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B      (A2)+,40*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B      (A2)+,40*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B      (A2)+,40*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B      (A2)+,40*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.L      #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L      #1,PuntaTesto        ; prossimo carattere da stampare

FineTesto:
        RTS


PuntaTesto:
        dc.l        TESTO

PuntaBitplane:
        dc.l        BufferVuoto+40*3

;        $00 per "fine linea" - $FF per "fine testo"

               ; numero caratteri per linea: 40
TESTO:            ;                  1111111111222222222233333333334
            ;   12345678901234567890123456789012345678901234567890
        dc.b        '                                        ',0 ; 1
        dc.b        '    Usa il mouse per spostare la        ',0 ; 2
        dc.b        '                                        ',0 ; 3
        dc.b        '    barra, cambiarla di colore,         ',0 ; 4
        dc.b        '                                        ',0 ; 5
        dc.b        '    suonare la musica o "chiudere"      ',0 ; 6
        dc.b        '                                        ',0 ; 7
        dc.b        '    lo schermo con il DIWSTART/STOP     ',$FF ; 12

        EVEN

;        Il FONT caratteri 8x8 copiato in CHIP dalla CPU e non dal blitter,
;        per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin      "assembler2:sorgenti4/nice.fnt"

*****************************************************************************
```

```
*                       ROUTINE MUSICALE
*****************************************************************************

        include         "music.s"

*****************************************************************************
;                       MEGACOPPERLISTONA GALATTICA (quasi...)
*****************************************************************************


        SECTION         GRAPHIC,DATA_C


COPPERLIST:
SpritePointers:
        dc.w        $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w        $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w        $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w        $13e,0

        dc.w        $8E         ; DiwStrt
DiwYStart:
        dc.b        $30
DIWXSTART:
        dc.b        $81
        dc.w        $90         ; DiwStop
DIWYSTOP:
        dc.b        $2c
DIWXSTOP:
        dc.b        $c1
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24                ; BplCon2
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%0100001000000000        ; BPLCON0 - 4 planes lowres (16 colori)

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane
        dc.w $e4,0,$e6,0        ;secondo bitplane
        dc.w $e8,0,$ea,0        ;terzo         bitplane
        dc.w $ec,0,$ee,0        ;quarto         bitplane

; i primi 16 colori sono per il LOGO

        dc.w $180,$000,$182,$fff,$184,$200,$186,$310
        dc.w $188,$410,$18a,$620,$18c,$841,$18e,$a73
        dc.w $190,$b95,$192,$db6,$194,$dc7,$196,$111
        dc.w $198,$222,$19a,$334,$19c,$99b,$19e,$446


        dc.w        $1A2,$fff        ; color17   Colore
        dc.w        $1A4,$fa6        ; color18   del
        dc.w        $1A6,$000        ; color19   mouse

BARRA:
        dc.w        $5c07,$FFFE        ; aspetto la linea $50
```

```
        dc.w        $180,$300       ; inizio la barra rossa: rosso a 3
        dc.w        $5d07,$FFFE       ; linea seguente
        dc.w        $180,$600       ; rosso a 6
        dc.w        $5e07,$FFFE
        dc.w        $180,$900       ; rosso a 9
        dc.w        $5f07,$FFFE
        dc.w        $180,$c00       ; rosso a 12
        dc.w        $6007,$FFFE
        dc.w        $180,$f00       ; rosso a 15 (al massimo)
        dc.w        $6107,$FFFE
        dc.w        $180,$c00       ; rosso a 12
        dc.w        $6207,$FFFE
        dc.w        $180,$900       ; rosso a 9
        dc.w        $6307,$FFFE
        dc.w        $180,$600       ; rosso a 6
        dc.w        $6407,$FFFE
        dc.w        $180,$300       ; rosso a 3
        dc.w        $6507,$FFFE
        dc.w        $180,$000       ; colore NERO


; barra rossa sotto il logo

        dc.w        $8407,$fffe       ; fine del logo

BPLPOINTER2:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $100,$1200       ; 1 bitplane (azzerato)

        dc.w        $8507,$FFFE       ; linea seguente
        dc.w        $180,$606       ; viola
        dc.w        $8607,$FFFE
        dc.w        $180,$909       ; viola
        dc.w        $8707,$FFFE
        dc.w        $180,$c0c       ; viola
        dc.w        $8807,$FFFE
        dc.w        $180,$f0f       ; viola (al massimo)
        dc.w        $8907,$FFFE
        dc.w        $180,$c0c       ; viola
        dc.w        $8a07,$FFFE
        dc.w        $180,$909       ; viola
        dc.w        $8b07,$FFFE
        dc.w        $180,$606       ; viola
        dc.w        $8c07,$FFFE
        dc.w        $180,$303       ; viola
        dc.w        $8d07,$FFFE
        dc.w        $180,$000       ; colore NERO

        dc.w        $182,$fe3       ; Colore testo

; barrettone centrale

        dc.w        $9007,$FFFE       ; linea seguente

        dc.w        $180,$011       ; celestino a 11
        dc.w        $9507,$FFFE
        dc.w        $180,$022       ; celestino a 22
        dc.w        $9a07,$FFFE
        dc.w        $180,$033       ; celestino a 33
        dc.w        $9f07,$FFFE
        dc.w        $180,$055       ; celestino a 55
```

```
        dc.w        $a407,$FFFE
        dc.w        $180,$077       ; celestino a 77
        dc.w        $a907,$FFFE
        dc.w        $180,$099       ; celestino a 99
        dc.w        $ae07,$FFFE
        dc.w        $180,$077       ; celestino a 77
        dc.w        $b307,$FFFE
        dc.w        $180,$055       ; celestino a 55
        dc.w        $b807,$FFFE
        dc.w        $180,$033       ; celestino a 33
        dc.w        $bd07,$FFFE
        dc.w        $180,$022       ; celestino a 22
        dc.w        $c207,$FFFE
        dc.w        $180,$011       ; celestino a 11

*****Figura di base:

        dc.w        $c607,$FFFE       ; Aspettiamo la linea c6
        dc.w        $180,$000       ; colore NERO

                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane sempre LoRes

BPLPOINTERSbase:
        dc.w $e0,$0000,$e2,$0000
CopBase:
        dc.w $0180,$0000,$0182,$0877

; barra rossa sopra il pannello

        dc.w        $ca07,$FFFE       ; linea seguente
        dc.w        $180,$606       ; rosso
        dc.w        $cb07,$FFFE
        dc.w        $180,$909       ; rosso
        dc.w        $cc07,$FFFE
        dc.w        $180,$c0c       ; rosso
        dc.w        $cd07,$FFFE
        dc.w        $180,$f0f       ; rosso (al massimo)
        dc.w        $ce07,$FFFE
        dc.w        $180,$c0c       ; rosso
        dc.w        $cf07,$FFFE
        dc.w        $180,$909       ; rosso
        dc.w        $d007,$FFFE
        dc.w        $180,$606       ; rosso
        dc.w        $d107,$FFFE
        dc.w        $180,$303       ; rosso
        dc.w        $d207,$FFFE
        dc.w        $180,$000       ; colore NERO

        dc.w        $ca07,$FFFE       ; WAIT - Aspetto la linea $ca
        dc.w        $180,$001       ; COLOR0 - blu scurissimo
        dc.w        $cc07,$FFFE       ; WAIT - linea 74 ($4a)
        dc.w        $180,$002       ; blu un po' piu' intenso
        dc.w        $ce07,$FFFE       ; linea 75 ($4b)
        dc.w        $180,$003       ; blu a 3
        dc.w        $d007,$FFFE       ; prossima linea
        dc.w        $180,$004       ; blu a 4
        dc.w        $d207,$FFFE       ; prossima linea
        dc.w        $180,$005       ; blu a 5
        dc.w        $d407,$FFFE       ; prossima linea
        dc.w        $180,$006       ; blu a 6
        dc.w        $d607,$FFFE       ; salto 2 linee: da $4e a $50, ossia da 78 a 80
```

```
        dc.w        $180,$007       ; blu a 7
        dc.w        $d807,$FFFE       ; sato 2 linee
        dc.w        $180,$008       ; blu a 8
        dc.w        $da07,$FFFE       ; salto 3 linee
        dc.w        $180,$009       ; blu a 9
        dc.w        $e007,$FFFE       ; salto 3 linee
        dc.w        $180,$00a       ; blu a 10
        dc.w        $e507,$FFFE       ; salto 3 linee
        dc.w        $180,$00b       ; blu a 11
        dc.w        $ea07,$FFFE       ; salto 3 linee
        dc.w        $180,$00c       ; blu a 12
        dc.w        $f007,$FFFE       ; salto 4 linee
        dc.w        $180,$00d       ; blu a 13
        dc.w        $f507,$FFFE       ; salto 5 linee
        dc.w        $180,$00e       ; blu a 14
        dc.w        $fa07,$FFFE       ; salto 6 linee
        dc.w        $180,$00f       ; blu a 15

        dc.w        $ffdf,$FFFE       ; aspetta linea $ff

        dc.w        $0207,$FFFE       ; aspetto
        dc.w        $182,$0f0       ; colore 1 verde

        dc.w        $0f07,$FFFE       ; aspetto
        dc.w        $182,$f22       ; colore 1 rosso

        dc.w        $1c07,$FFFE       ; aspetto
        dc.w        $182,$ff0       ; colore 1 giallo

        dc.w        $2907,$FFFE       ; aspetto
        dc.w        $182,$877       ; colore 1 grigio

        dc.w        $FFFF,$FFFE       ; Fine della copperlist

*****************************************************************************
*                               Sprite                                      *
*****************************************************************************
; Come sempre, la grafica va SOLO caricata in CHIP come la Copperlist!!

MIOSPRITE0:
VSTART0:
        dc.b $50
HSTART0:
        dc.b $45
VSTOP0:
        dc.b $5d
VHBITS0:
        dc.b $00
 dc.w       %0110000000000000,%1000000000000000
 dc.w       %0001100000000000,%1110000000000000
 dc.w       %1000011000000000,%1111100000000000
 dc.w       %1000000110000000,%1111111000000000
 dc.w       %0100000000000000,%0111111110000000
 dc.w       %0100000000000000,%0111111000000000
 dc.w       %0010000100000000,%0011111000000000
 dc.w       %0010010010000000,%0011111100000000
 dc.w       %0001001001000000,%0001101110000000
 dc.w       %0001000100100000,%0001100111000000
 dc.w       %0000000010000000,%0000000011100000
 dc.w       %0000000000000000,%0000000000000000
 dc.w       %0000000000000000,%0000000000000000
 dc.w       0,0
```

```
SpriteNullo:                        ; Sprite nullo da puntare in copperlist
        dc.l        0,0,0,0                 ; negli eventuali puntatori inutilizzati

PICTUREbase:
        incbin        "base320*105*1.raw"

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).

PICTURE1:
        incbin        "logo320*84*16c.raw"


; Musica. Attenzione: la routine "music.s" del disco 2 non e' la stessa di
; quella del disco 1. Le 2 modifiche sono la rimossione di un BUG che alle
; volte causava una guru all'uscita del programma, e il fatto che in mt_data
; e' un puntatore alla musica, e non LA musica. Questo permette di cambiare
; la musica piu' facilmente.

mt_data:
        dc.l        mt_data1

mt_data1:
        incbin        "mod.JamInExcess"

        Section        MiniBitplane,bss_c

;        In questo buffer viene stampato il testo

BufferVuoto:
        ds.b        40*68

        end                        ; Il Computer non legge oltre l' END!
                                ; Adesso possiamo scrivere qualsiasi cosa senza
                                ; i PUNTI e VIRGOLA o ASTERISCHI
```

Volendo un effetto video diverso ad ogni "tasto premuto", dovremo sapere se il tasto sinistro e' premuto e, in tal caso, sapere la posizione dello sprite del mouse. In poche parole dovremo sapere quale tasto è stato premuto per eseguire un diverso effetto-video:

Appena partiamo col programma troviamo un controllo: 'Tasto sinistro premuto?', se il tasto non e' stato premuto continueremo col programma aggiornando la posizione del mouse, muovendo la freccia per lo schermo, se invece e' stato premuto, saltiamo ad una routine che compara la posizione di:
- Sprite_x
- Sprite_y

con le coordinate dove si trovano i nostri tasti!

********** Trucchetto del mestiere **********

Ma come si fa a sapere le coordinate X ed Y dei nostri "bottoni"? tranquilli, non dovete fare miliardi di prove o calcoli ad occhio! Dato che ASMONE ha un monitor L.M. incorporato, possiamo fare in questo modo: disegnatevi il pannello di controllo che volete, con i vostri bottoncini; una volta puntato e visualizzato il tutto, con la routine del mouse, e' giusto il momento di sapere a quali coordinate corrispondono i bottoni.

Se volete verificare la posizione di ogni tasto basta mettere all'inizio del programma (al posto di ****1), questo semplice loop:

```
Aspetta:
        bsr.w        LeggiMouse
        move.w        sprite_y(pc),d0
        move.w        sprite_x(pc),d1
        lea         miosprite0,a1
        moveq         #13,d2
        bsr.w         UniMuoviSprite
        btst         #2,$dff016
        bne.w         Aspetta
        bra.w         esci
```

il quale aggiorna la posizione del mouse, lo muove e, quando viene premuto il tasto sinistro del mouse, usciamo semplicemente dal programma!!
Posizionatevi alla coordinata che volete sapere, ad esempio un angolo di un bottone, e uscite col tasto sinistro rimanendo in quel punto.
Ora bastera' solamente vedere le ultime posizioni assunte dal mouse con il mitico comando "M" (dopo aver premuto il tasto ESC):

```
        m Sprite_x   (premere RETURN)
        m Sprite_y   (premere RETURN)
```

il comando "M" e' utilissimo. Viene usato molto per verificare a che "punto" o con quale "valore" si e' arrivati. Per esempio se voleste far fermare uno sprite o una barra ad un certo punto, basta fare un loop che lo fa avanzare fino a che non si preme il mouse. Lanciate il programma, premete il mouse quando e' arrivato al punto che volete, e fare "M variabile". Semplice!!!

****

Come prova, provate a far apparire lo sprite in posti diversi dello schermo al momento dell'avviamento del programma, inoltre provate a far restare il puntatore del mouse nel rettangolo della figura inferiore.

Una cosa che avrete senz'altro notato, è il fatto che se premiamo il tasto "+" o "-", e non rilasciamo il tasto sinistro del mouse, la barra continua imperterrita a muoversi anche se spostiamo la freccia fuori dal bottone: questo perchè, come spiegato nel punto **2, fino a che non abbiamo rilasciato il tasto del mouse, il programma non riverifica la posizione del mouse!
Per modificare questo fatto, basta aggiungere al punto **2:

```
        bsr.s        MuoviFreccia
```

ommettendo ovviamente il

```
        btst.b        #6,$bfe001
        beq.s        Piu
```

Provate anche a cambiare il punto **3

Adesso aggiungete solo :

```
        brs.s        MuoviFreccia
```

Ai punti **4, **5, e guardate cosa succede:
basta entrare o uscire dal tasto che parte il suo effetto!

A differenza degli altri tasti, per quelli "cambia colore a barra", basta solamente passarci sopra col mouse premuto per avere l'effeto desiderato!
Adesso dovreste essere capaci di sapere il perche'!

Infine, I tasti che attivano e disattivano la musica "bloccano" anche la

freccia.... a voi l'arduo problema di capire perche'.

Lezione8h2

```
; Lezione8h2.s                Routine di scrolltext 8*8, che usa solo il bplcon1
;                         per scorrere. Originale di Lorenzo Di Gaetano.

        SECTION        SysInfo,CODE

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA

START:

;        Puntiamo i bitplanes in copperlist

        MOVE.L        #schermo,d0        ; in d0 mettiamo l'indirizzo del bitplane
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                    ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        clr.w        ContaScroll        ; Azzera il contatore dello scroll
        bsr.w        Print                ; Stampa la prima volta

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$12c00,d2        ; linea da aspettare = $12c
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $12c
        BNE.S        Waity1
Waity2:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $12c
        Beq.S        Waity2

        bsr.s        Scroll                ; Routine sche scrolla a sinistra il testo
                                ; con bplcon1, e ogni 16 pixel lo ristampa
                                ; 2 caratteri (8*2=16 pixel) piu' avanti
                                ; resettando il bplcon1 -> SCROLLING!!!

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
        rts
```

```
*****************************************************************************
; Routine che decide se scrollare col bplcon1, o ristampando l'intero testo
; 2 caratteri (16 pixel) piu' a sinistra (questo 1 volta ogni 16, naturalmente)
*****************************************************************************

Scroll:
        tst.b        Scrolling        ; Abbiamo scrollato al massimo con bplcon1?
        bne.s        AlloraAdda        ; Se non ancora, continua la sottrazione

; Altrimenti ripartiamo da $FF e ristampiamo il testo 2 caratteri avanti!

        addq.w        #2,ContaScroll        ; 2 caratteri piu' avanti -> 16 pixel avanti
        move.b        #$FF,Scrolling        ; Resetta bplcon1
        bsr.s        Print                ; e ristampa il ScrollText 2 caratteri piu'
        rts                        ; avanti,  ossia 2*8=16 pixel piu' avanti.

AlloraAdda:
        sub.b        #$11,Scrolling        ; 1 pixel verso sinistra con bplcon1
        rts

*****************************************************************************
; Routine di print 8*8 modificata per lo scroll
*****************************************************************************

Print:
        lea        Schermo+(42*192),a0        ; Indirizzo dove stampare
        lea        ScrollText(PC),a1        ; Indirizzo scrolltext (ascii)
        moveq        #42-1,d2                ; Numero caratteri da stampare
        moveq        #0,d0
        move.w        ContaScroll(PC),d0        ; Offset dall'inizio del ScrollText
        add.l        d0,a1                        ; Trova il carattere nello scrolltext
Printriga:
        sub.l        a2,a2                ; azzera a2
        moveq        #0,d1
        move.b        (a1)+,d1
        cmp.b        #$ff,d1                ; flag di fine ScrollText?
        bne.s        NonRipartire        ; Se non ancora, continua
        clr.w        Contascroll        ; Oppure, riparti dall'inizio del ScrollText
NonRipartire:
        sub.b        #$20,d1
        lsl.w        #3,d1                ; moltiplica per 8
        move.l        d1,a2
        add.l        #Fonts,a2        ; trova il carattere nel font
        move.b        (a2)+,(a0)
        move.b        (a2)+,42(a0)        ; 42 per compensare il dfstart e andare quindi
        move.b        (a2)+,42*2(a0)        ; oltre lo schermo
        move.b        (a2)+,42*3(a0)
        move.b        (a2)+,42*4(a0)
        move.b        (a2)+,42*5(a0)
        move.b        (a2)+,42*6(a0)
        move.b        (a2)+,42*7(a0)
        addq.w        #1,a0                ; prossimo carattere
        dbra        d2,Printriga
        rts

ContaScroll:
        dc.w        0


ScrollText:
```

```
        dc.b        "                                              "
        dc.b        "QUESTO TESTO VIENE SPOSTATO CON IL REGISTRO BPLCON1:"
        DC.B        " DOPO AVERLO SPOSTATO DI 16 PIXEL VIENE AZZERATO,"
        DC.B        " E INVECE DI PUNTARE ALLA PROSSIMA WORD DELL‘ IMMAGINE IL TE"
        DC.B        "STO VIENE RISTAMPATO SULLO SCHERMO 2 LETTERE DOPO."
        DC.B        "L’AUTORE, LORENZO DI GAETANO, (The Amiga Dj) HA FATTO QUESTA"
        dc.b        " ROUTINE CON LE SOLE CONOSCENZE DEL DISCO 1 DEL CORSO."
        dc.b        "... FORZA AMIGA!!!                    "
        DC.B        "                                       "
        dc.b        $FF        ; Flag di fine scrolltext

        even

; Font 8x8

Fonts:
        incbin        "nice.fnt"

;***************************************************************************

        SECTION        GRAPHIC,DATA_C


COPPERLIST:
        dc.w    $08e,$2c81       ; Qui‘ ci sono i registri standard
        dc.w    $090,$2cc1
        dc.w    $092,$0038
        dc.w    $094,$00d0
        dc.w        $102,0
        dc.w    $104,0
        dc.w    $108,2                        ;2 per saltare il vuoto dell‘ immagine
        dc.w    $10a,2

bplpointers:
        dc.w    $e0,$0000,$e2,$0000    ; Puntatori ai bitplane

        dc.w    $100,%0001001000000000 ; Bplcon0 2 colori

        dc.w        $180,$000
        dc.w        $182,$888

; Qua potrebbe starci un’immagine qualsiasi...

        dc.w        $eb07,$fffe        ; Qui‘ comincia la copperlist dello
                                ; scrolling.
        dc.w    $092,$0030        ; Per nascondere l‘errore di scroll
        dc.w    $094,$00d0

        dc.w    $104,0
        dc.w    $108,0
        dc.w    $10a,0
        dc.w    $102                ; bplcon1
        dc.b        $00
Scrolling:
        dc.b        $FF
        dc.w        $182,$200
        dc.w        $ec07,$FFFe
        dc.w        $182,$400
        dc.w        $ed07,$fffe
        dc.w        $182,$600
        dc.w        $ee07,$fffe
        dc.w        $182,$800
```

```
        dc.w        $ef07,$fffe
        dc.w        $182,$a00
        dc.w        $f007,$fffe
        dc.w        $182,$d00
        dc.w        $f107,$fffe
        dc.w        $182,$a00
        dc.w        $f207,$fffe
        dc.w        $182,$800
        dc.w        $f307,$fffe

        dc.w        $182,$000       ; Effetti copper...
        dc.w        $180,$001
        dc.w        $108,-84
        dc.w        $10a,-84
        dc.w        $f4ff,$fffe
        dc.w        $180,$003
        dc.w        $f5ff,$fffe
        dc.w        $180,$005
        dc.w        $f6ff,$fffe
        dc.w        $180,$007
        dc.w        $f7ff,$fffe
        dc.w        $180,$009
        dc.w        $f8ff,$fffe
        dc.w        $180,$00b
        dc.w        $f8ff,$fffe
        dc.w        $180,$00c
        dc.w        $f9ff,$fffe
        dc.w        $180,$00f
        dc.w        $faff,$fffe
        dc.w        $180,$00f
        dc.w        $fbff,$fffe
        dc.w        $180,$000
        dc.w        $108,0
        dc.w        $10a,0
        dc.w        $ffff,$fffe       ; Fine copperlist

;***************************************************************************

        Section        Bitplanozzo,bss_C

Schermo:
        ds.b        42*256

        end
```

24.9 Lezione8i

```
; Lezione8i - Semplici equalizzatori temporizzati con la routine musicale
;            - TASTO DESTRO per cambiare velocita' delle barre

        SECTION        MAINPROGRAM,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
***************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001010000000        ; solo copper DMA
```

```
;                -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA          (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
        MOVE.W        #DMASET,$96(a5)               ; DMACON - abilita bitplane, copper
        move.l        #MyCopList,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        bsr.w        mt_init        ; Inizializza la routine musicale

MainLoop:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

        bsr.w        mt_music                ; suona la musica

        btst        #2,$dff016        ; tasto destro premuto?
        beq.s        VaiForte
        move.b        #2,EqualSpeed        ; velocita' di calo = 2 pixel a frame
        bra.s        VaiPiano
Vaiforte:
        move.b        #8,EqualSpeed        ; velocita' di calo = 8 pixel a frame
VaiPiano:

        bsr.s        Equalizzatori                ; Semplice routine equalizzatori

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001                ; LMB premuto?
        bne.s        MainLoop                ; Se "NO" ricomincia

        bsr.w        mt_end        ; ferma la routine musicale
        rts


; Ecco la routine degli equalizzatori, l'audio analizer. La prima cosa da
; sapere e' dove trovare informazioni sull'utilizzo delle 4 voci da parte
; della routine musicale. Solitamente si usa controllare la variabile della
; replay routine che ci puo' segnalare se viene attivata una voce per suonare
; uno strumento, solitamente "mt_chanXtemp", dove la X puo' essere 1,2,3 o 4.
; Questo sistema pero' non e' la perfezione, dato che possiamo sapere soltanto
```

```
; quando "comuncia" l'utilizzo di una delle 4 voci, per cui se per esempio
; viene suonato uno strumento che continua a suonare per 10 secondi, la
; barretta di quella voce si allunghera' al primo secondo, segnalando l'uso
; di quella voce in quel momento, ma poi si abbassera' e rimarra' tranquilla
; fino a che quella voce non sara' usata per suonare un'altro strumento.
; Se una voce e' usata per suoni corti, come la batteria, non si nota questo
; fatto, dato che al momento del BUM! la barretta sale, e quando e' scesa
; nuovamente il suono e' finito o sta per finire. Il problema diviene tragico
; quando vuene usato uno strumento che fa il "loop", ad esempio le voices,
; per cui la barretta fa un solo "saltello", poi rimane giu' durante il loop.
; Questo sistema e' lo stesso delle barre presenti sulle 4 tracce dei vecchi
; soundtracker e protracker fino alla versione 2. Per equalizzatori del tipo
; del protracker 3, che seguono piu' fedelmente il "volume" delle voci, occorre
; modificare la routine musicale stessa, lo stesso vale per gli equalizzatori
; che visualizzano la forma d'onda. Basta fare un "tst.w mt_chanXtemp", e si
; puo' agire di conseguenza. In questo caso vengono mosse delle barre fatte col
; copper, utilizzando la posizione orizzontale dei wait della copperlist, in
; questo modo usiamo solo il colore di sfondo, $dff180, senza bitplanes.
; Basta aspettare l'inizio della linea, mettere il colore della barra, e poi
; mettere un wait che aspetti una posizione orizzontale piu' avanzata, ma
; nella stessa linea, dopodiche' rimettere il colore dello sfondo. In questo
; modo, agendo su quel wait, "spostiamo" in avanti e indietro la barra, come
; abbiamo visto in lezione3g.s e Lezione3h.s
; La routine in pratica fa questo: ogni fotogramma cala le barre, fino a che
; non sono azzerate, per cui se non c'e' musica rimangono azzerate. Nel caso
; che il tst dell'mt_chanXtemp segnali un sample suonato in quella voce, mette
; alla corrispondenza barra il valore massimo, ossia $a7.


Equalizzatori:
        move.b        EqualSpeed(PC),d0 ; velocita' di "calo" delle barre in d0
        cmp.b        #$07,WaitEqu1+1        ; la prima barra e' calata a zero?
        bls.s        NonAbbass1        ; se si, non la abbassare ulteriormente!
                                ; * bls significa minore o uguale, e' meglio
                                ; usarlo al posto del beq perche' sottraendo
                                ; con d0 un numero troppo grande puo'
                                ; succedere che si vada a $05 o $03!
        sub.b        d0,WaitEqu1+1        ; altrimenti abbassa la barra, composta da
        sub.b        d0,WaitEqu1b+1        ; due linee colorate e una nera
        sub.b        d0,WaitEqu1c+1
NonAbbass1:
        tst.w        mt_chan1temp        ; voce 1 non "suonata"?
        beq.s        anal2                ; se no, salta ad Anal2
        clr.w        mt_chan1temp        ; azzera per aspettare la prossima scrittura
        move.b        #$a7,WaitEqu1+1        ; BARRA AL MASSIMO!
        move.b        #$a7,WaitEqu1b+1
        move.b        #$a7,WaitEqu1c+1
anal2:
        cmp.b        #$07,WaitEqu2+1        ; la seconda barra e' calata a zero?
        bls.s        NonAbbass2        ; se si, non la abbassare ulteriormente!
        sub.b        d0,WaitEqu2+1        ; altrimenti abbassa la barra
        sub.b        d0,WaitEqu2b+1
        sub.b        d0,WaitEqu2c+1
NonAbbass2:
        tst.w        mt_chan2temp        ; voce 2 non "suonata"?
        beq.s        anal3                ; se no, salta ad Anal3
        clr.w        mt_chan2temp        ; azzera per aspettare la prossima scrittura
        move.b        #$a7,WaitEqu2+1        ; BARRA AL MASSIMO!
        move.b        #$a7,WaitEqu2b+1
        move.b        #$a7,WaitEqu2c+1
anal3:
        cmp.b        #$07,WaitEqu3+1        ; la terza barra e' calata a zero?
```

```
        bls.s       NonAbbass3        ; se si, non la abbassare ulteriormente!
        sub.b       d0,WaitEqu3+1       ; altrimenti abbassa la barra
        sub.b       d0,WaitEqu3b+1
        sub.b       d0,WaitEqu3c+1
NonAbbass3:
        tst.w       mt_chan3temp        ; voce 3 non "suonata"?
        beq.s       anal4                ; se no, salta ad Anal4
        clr.w       mt_chan3temp        ; azzera per aspettare la prossima scrittura
        move.b       #$a7,WaitEqu3+1        ; BARRA AL MASSIMO!
        move.b       #$a7,WaitEqu3b+1
        move.b       #$a7,WaitEqu3c+1
anal4:
        cmp.b       #$07,WaitEqu4+1        ; la quarta barra e' calata a zero?
        bls.s       NonAbbass4        ; se si, non la abbassare ulteriormente!
        sub.b       d0,WaitEqu4+1       ; altrimenti abbassa la barra
        sub.b       d0,WaitEqu4b+1
        sub.b       d0,WaitEqu4c+1
NonAbbass4:
        tst.w       mt_chan4temp        ; voce 4 non "suonata"?
        beq.s       analizerend        ; se no, esci!
        clr.w        mt_chan4temp        ; azzera per aspettare la prossima scrittura
        move.b       #$a7,WaitEqu4+1        ; BARRA AL MASSIMO!
        move.b       #$a7,WaitEqu4b+1
        move.b       #$a7,WaitEqu4c+1
analizerend:
        rts

EqualSpeed:
        dc.b        4
        even

*****************************************************************************
;        ROUTINE MUSICALE

        include        "music.s"
*****************************************************************************

        Section        DatiChippy,data_C

MyCopList:
        dc.w        $100,$200        ; Bplcon0 - no bitplanes
        dc.w        $180,$00e        ; color0 blu
        dc.w        $ffdf,$fffe        ; aspetta la linea $FF

;        wait&mode della routine analyzer - usano la posizione orizzontale
;        dei wait per far andare in "avanti" e "indietro" le barre

        dc.w        $1507,$fffe        ; wait inizio riga
        dc.w        $180,$00e        ; color0 blu

        dc.w        $1607,$fffe        ; wait inizio riga
        dc.w        $180,$f55        ; color0 ROSSO - colore prima BARRA
WaitEqu1:
        dc.w        $1617,$fffe        ; wait (sara' modificato come fine riga, poi
                                ; calera' di 4 in 4 fino a tornare a $07)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $1707,$fffe        ; wait inizio riga
        dc.w        $180,$f55        ; color0 ROSSO (barra alta 2 linee!)
WaitEqu1b:
        dc.w        $1717,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $1807,$fffe        ; wait inizio riga
```

```
        dc.w        $180,$002       ; color0 NERO ("ombra" sotto la prima barra)
WaitEqu1c:
        dc.w        $1817,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu

; seconda barra

        dc.w        $1b07,$fffe       ; wait inizio linea
        dc.w        $180,$a5f       ; color0 VIOLA (seconda BARRA)
WaitEqu2:
        dc.w        $1b17,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu
        dc.w        $1c07,$fffe       ; wait inizio linea
        dc.w        $180,$a5f       ; colore SECONDA BARRA (alta 2 linee!)
WaitEqu2b:
        dc.w        $1c17,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu
        dc.w        $1d07,$fffe       ; wait inizio linea
        dc.w        $180,$002       ; color0 nero ("ombra")
WaitEqu2c:
        dc.w        $1d17,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu

; terza barra

        dc.w        $2007,$fffe       ; wait inizio linea
        dc.w        $180,$ff0       ; colore TERZA BARRA
WaitEqu3:
        dc.w        $2017,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu
        dc.w        $2107,$fffe       ; wait inizio linea
        dc.w        $180,$ff0       ; colore TERZA BARRA (alta 2 linee!)
WaitEqu3b:
        dc.w        $2117,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu
        dc.w        $2207,$fffe       ; wait inizio linea
        dc.w        $180,$002       ; color0 nero ("ombra")
WaitEqu3c:
        dc.w        $2217,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu

; quarta barra

        dc.w        $2507,$fffe       ; wait inizio linea
        dc.w        $180,$5F0       ; colore QUARTA BARRA
WaitEqu4:
        dc.w         $2517,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu
        dc.w        $2607,$fffe       ; wait inizio linea
        dc.w        $180,$5F0       ; colore QUARTA BARRA (alta 2 linee!)
WaitEqu4b:
        dc.w         $2617,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu
        dc.w        $2707,$fffe       ; wait inizio linea
        dc.w        $180,$002       ; color0 nero ("ombra")
WaitEqu4c:
        dc.w         $2717,$fffe       ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e       ; color0 blu

        DC.W        $FFFF,$FFFE       ; fine copperlist
```

```
; Musica. Attenzione: la routine "music.s" del disco 2 non e' la stessa di
; quella del disco 1. Le 2 modifiche sono la rimossione di un BUG che alle
; volte causava una guru all'uscita del programma, e il fatto che mt_data
; e' un puntatore alla musica, e non LA musica. Questo permette di cambiare
; la musica piu' facilmente.

; potete scegliere una delle 4 musichette presenti nel disco.

mt_data:
        dc.l        mt_data1

Mt_data1:
;        incbin        "mod.fairlight"                ; by d-zire/silents 92 (lungo solo 2k!)
        incbin        "mod.fuck the bass"        ; by m.c.m/remedy 91
;        incbin        "mod.yellowcandy"        ; by sire/supplex
;        incbin        "mod.JamInexcess"        ; by raiser/ram jam

        end

Potete usare questo sorgente per sentire le 4 musichette protracker di questo
disco. Il "mod.fairlight" e' una delle musiche piu' "sintetiche" possibili,
infatti e' lunga solamente 2374 bytes, e compattata col PowerPacker diventa
lunga 952 bytes!!!
```

Lezione8i2

```
; Lezione8i2 - Semplici equalizzatori temporizzati con la routine musicale
;            - TASTO DESTRO per cambiare velocita' delle barre

        SECTION        MAINPROGRAM,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001010000000        ; solo copper DMA
;                 -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
        move.l        #MyCopList,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        bsr.w        mt_init        ; Inizializza la routine musicale

MainLoop:
```

```
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        bsr.w        mt_music        ; suona la musica

        btst        #2,$dff016        ; tasto destro premuto?
        beq.s        Uscita                ; se si, esci

        bsr.s        Equalizzatori        ; Semplice routine equalizzatori

        btst        #6,$bfe001        ; LMB premuto?
        bne.s        MainLoop        ; Se "NO" ricomincia, altrimenti cambia musica

        lea        musiche(PC),a0                ; scambia le musiche
        move.l        (a0),d0
        move.l        4(a0),d1
        move.l        4*2(a0),d2
        move.l        4*3(a0),d3
        move.l        d0,4(a0)
        move.l        d1,4*2(a0)
        move.l        d2,4*3(a0)
        move.l        d3,(a0)

        move.l        musiche(PC),mt_data        ; punta la musica attuale
        bsr.w        mt_init                        ; resetta la musica
SpettaLascia:
        btst        #6,$bfe001                ; LMB sempre premuto?
        beq.s        SpettaLascia                ; aspetta che sia lasciato
        bra.s        MainLoop

Uscita:
        bsr.w        mt_end        ; ferma la routine musicale
        rts

; Tabella con le musiche... ruotando gli indirizzi e copiando sempre il
; primo possiamo cambiarle...

musiche:
        dc.l        mt_data1,mt_data2,mt_data3,mt_data4


; Ecco la routine degli equalizzatori, l'audio analizer. La prima cosa da
; sapere e' dove trovare informazioni sull'utilizzo delle 4 voci da parte
; della routine musicale. Solitamente si usa controllare la variabile della
; replay routine che ci puo' segnalare se viene attivata una voce per suonare
; uno strumento, solitamente "mt_chanXtemp", dove la X puo' essere 1,2,3 o 4.
; In questa versione, anche il valore che ha mt_chanXtemp viene usato per
; analyzer supplementari.


Equalizzatori:
        move.b        EqualSpeed(PC),d0 ; velocita' di "calo" delle barre in d0
```

```
        cmp.b       #$07,WaitEqu1+1        ; la prima barra e' calata a zero?
        bls.s       NonAbbass1        ; se si, non la abbassare ulteriormente!
                            ; * bls significa minore o uguale, e' meglio
                            ; usarlo al posto del beq perche' sottraendo
                            ; con d0 un numero troppo grande puo'
                            ; succedere che si vada a $05 o $03!
        sub.b       d0,WaitEqu1+1         ; altrimenti abbassa la barra, composta da
        sub.b       d0,WaitEqu1b+1         ; due linee colorate e una nera
        sub.b       d0,WaitEqu1c+1
NonAbbass1:
        tst.w       mt_chan1temp        ; voce 1 non "suonata"?
        beq.s       anal2                ; se no, salta ad Anal2
        move.w       mt_chan1temp,COLORE1
        move.w       mt_chan1temp,COLORE1b
        and.w       #$f0,COLORE1        ; seleziona solo componente blu
        ori.w       #$330,COLORE1        ; come minimo $30!
        and.w       #$0f,COLORE1b        ; seleziona solo componente verde
        ori.w       #$303,COLORE1b        ; come minimo $03!
        clr.w       mt_chan1temp        ; azzera per aspettare la prossima scrittura
        move.b       #$a7,WaitEqu1+1        ; BARRA AL MASSIMO!
        move.b       #$a7,WaitEqu1b+1
        move.b       #$a7,WaitEqu1c+1
anal2:
        cmp.b       #$07,WaitEqu2+1        ; la seconda barra e' calata a zero?
        bls.s       NonAbbass2        ; se si, non la abbassare ulteriormente!
        sub.b       d0,WaitEqu2+1        ; altrimenti abbassa la barra
        sub.b       d0,WaitEqu2b+1
        sub.b       d0,WaitEqu2c+1
NonAbbass2:
        tst.w       mt_chan2temp        ; voce 2 non "suonata"?
        beq.s       anal3                ; se no, salta ad Anal3
        move.w       mt_chan2temp,COLORE2
        move.w       mt_chan2temp,COLORE2b
        and.w       #$f0,COLORE2        ; seleziona solo componente blu
        ori.w       #$330,COLORE2        ; come minimo $30!
        and.w       #$0f,COLORE2b        ; seleziona solo componente verde
        ori.w       #$303,COLORE2b        ; come minimo $03!
        clr.w       mt_chan2temp        ; azzera per aspettare la prossima scrittura
        move.b       #$a7,WaitEqu2+1        ; BARRA AL MASSIMO!
        move.b       #$a7,WaitEqu2b+1
        move.b       #$a7,WaitEqu2c+1
anal3:
        cmp.b       #$07,WaitEqu3+1        ; la terza barra e' calata a zero?
        bls.s       NonAbbass3        ; se si, non la abbassare ulteriormente!
        sub.b       d0,WaitEqu3+1        ; altrimenti abbassa la barra
        sub.b       d0,WaitEqu3b+1
        sub.b       d0,WaitEqu3c+1
NonAbbass3:
        tst.w       mt_chan3temp        ; voce 3 non "suonata"?
        beq.s       anal4                ; se no, salta ad Anal4
        move.w       mt_chan3temp,COLORE3
        move.w       mt_chan3temp,COLORE3b
        and.w       #$f0,COLORE3        ; seleziona solo componente blu
        ori.w       #$330,COLORE3        ; come minimo $30!
        and.w       #$0f,COLORE3b        ; seleziona solo componente verde
        ori.w       #$303,COLORE3b        ; come minimo $03!
        clr.w       mt_chan3temp        ; azzera per aspettare la prossima scrittura
        move.b       #$a7,WaitEqu3+1        ; BARRA AL MASSIMO!
        move.b       #$a7,WaitEqu3b+1
        move.b       #$a7,WaitEqu3c+1
anal4:
        cmp.b       #$07,WaitEqu4+1        ; la quarta barra e' calata a zero?
```

```
        bls.s       NonAbbass4        ; se si, non la abbassare ulteriormente!
        sub.b       d0,WaitEqu4+1       ; altrimenti abbassa la barra
        sub.b       d0,WaitEqu4b+1
        sub.b       d0,WaitEqu4c+1
NonAbbass4:
        tst.w       mt_chan4temp        ; voce 4 non "suonata"?
        beq.s       analizerend       ; se no, esci!
        move.w       mt_chan4temp,COLORE4
        move.w       mt_chan4temp,COLORE4b
        and.w       #$f0,COLORE4        ; seleziona solo componente blu
        ori.w       #$330,COLORE4        ; come minimo $30!
        and.w       #$0f,COLORE4b        ; seleziona solo componente verde
        ori.w       #$303,COLORE4b        ; come minimo $03!
        clr.w        mt_chan4temp        ; azzera per aspettare la prossima scrittura
        move.b       #$a7,WaitEqu4+1       ; BARRA AL MASSIMO!
        move.b       #$a7,WaitEqu4b+1
        move.b       #$a7,WaitEqu4c+1
analizerend:
        rts

EqualSpeed:
        dc.b        4
        even

*****************************************************************************
;        ROUTINE MUSICALE

        include        "music.s"
*****************************************************************************

        Section        DatiChippy,data_C

MyCopList:
        dc.w        $100,$200        ; Bplcon0 - no bitplanes
        dc.w        $180,$00e        ; color0 blu

        dc.w        $4807,$fffe
        dc.w        $180,$ddd
        dc.w        $4a07,$fffe
        dc.w        $180,$777

        dc.w        $5007,$fffe        ; wait inizio riga
        dc.w        $180
COLORE1:
        dc.w        $060
        dc.w        $5507,$fffe        ; wait inizio riga
        dc.w        $180
COLORE2:
        dc.w        $060
        dc.w        $5a07,$fffe        ; wait inizio riga
        dc.w        $180
COLORE3:
        dc.w        $060
        dc.w        $5f07,$fffe        ; wait inizio riga
        dc.w        $180
COLORE4:
        dc.w        $060
        dc.w        $6407,$fffe        ; wait inizio riga
        dc.w        $180
COLORE1b:
        dc.w        $00e
        dc.w        $6907,$fffe        ; wait inizio riga
```

```
        dc.w        $180
COLORE2b:
        dc.w        $00e
        dc.w        $6e07,$fffe        ; wait inizio riga
        dc.w        $180
COLORE3b:
        dc.w        $00e
        dc.w        $7307,$fffe        ; wait inizio riga
        dc.w        $180
COLORE4b:
        dc.w        $00e
        dc.w        $7807,$fffe        ; wait inizio riga
        dc.w        $180,$777
        dc.w        $7e07,$fffe
        dc.w        $180,$333
        dc.w        $8007,$fffe
        dc.w        $180,$00e

        dc.w        $ffdf,$fffe        ; aspetta la linea $FF

;        wait&mode della routine analyzer - usano la posizione orizzontale
;        dei wait per far andare in "avanti" e "indietro" le barre

        dc.w        $1507,$fffe        ; wait inizio riga
        dc.w        $180,$00e        ; color0 blu

        dc.w        $1607,$fffe        ; wait inizio riga
        dc.w        $180,$f55        ; color0 ROSSO - colore prima BARRA
WaitEqu1:
        dc.w        $1617,$fffe        ; wait (sara' modificato come fine riga, poi
                                ; calera' di 4 in 4 fino a tornare a $07)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $1707,$fffe        ; wait inizio riga
        dc.w        $180,$f55        ; color0 ROSSO (barra alta 2 linee!)
WaitEqu1b:
        dc.w        $1717,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $1807,$fffe        ; wait inizio riga
        dc.w        $180,$002        ; color0 NERO ("ombra" sotto la prima barra)
WaitEqu1c:
        dc.w        $1817,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu

; seconda barra

        dc.w        $1b07,$fffe        ; wait inizio linea
        dc.w        $180,$a5f        ; color0 VIOLA (seconda BARRA)
WaitEqu2:
        dc.w        $1b17,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $1c07,$fffe        ; wait inizio linea
        dc.w        $180,$a5f        ; colore SECONDA BARRA (alta 2 linee!)
WaitEqu2b:
        dc.w        $1c17,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $1d07,$fffe        ; wait inizio linea
        dc.w        $180,$002        ; color0 nero ("ombra")
WaitEqu2c:
        dc.w        $1d17,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu

; terza barra
```

```
        dc.w        $2007,$fffe        ; wait inizio linea
        dc.w        $180,$ff0        ; colore TERZA BARRA
WaitEqu3:
        dc.w        $2017,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $2107,$fffe        ; wait inizio linea
        dc.w        $180,$ff0        ; colore TERZA BARRA (alta 2 linee!)
WaitEqu3b:
        dc.w        $2117,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $2207,$fffe        ; wait inizio linea
        dc.w        $180,$002        ; color0 nero ("ombra")
WaitEqu3c:
        dc.w        $2217,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu

; quarta barra

        dc.w        $2507,$fffe        ; wait inizio linea
        dc.w        $180,$5F0        ; colore QUARTA BARRA
WaitEqu4:
        dc.w         $2517,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $2607,$fffe        ; wait inizio linea
        dc.w        $180,$5F0        ; colore QUARTA BARRA (alta 2 linee!)
WaitEqu4b:
        dc.w         $2617,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu
        dc.w        $2707,$fffe        ; wait inizio linea
        dc.w        $180,$002        ; color0 nero ("ombra")
WaitEqu4c:
        dc.w         $2717,$fffe        ; wait (modificato per lunghezza barra)
        dc.w        $180,$00e        ; color0 blu

        DC.W        $FFFF,$FFFE        ; fine copperlist


; musica - potete scegliere una delle 4 musichette presenti nel disco.
; qua si "capisce" l'utilita' dell'mt_data usato come puntatore.

mt_data:
        dc.l        mt_data1


mt_data1:
        incbin        "mod.fuck the bass"        ; by m.c.m/remedy 91
mt_data2:
        incbin        "mod.yellowcandy"        ; by sire/supplex
mt_data3:
        incbin        "mod.fairlight"                ; by d-zire/silents 92 (lungo solo 2k!)
mt_data4:
        incbin        "mod.JamInexcess"        ; by raiser/ram jam

        end
```

24.10 Lezione8l

```
; Lezione8l.s - Routine di stampa di punti (plot)
```

```
        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                 -----a-bcdefghij

START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        lea        bitplane,a0        ; Indirizzo del bitplane dove stampare

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

        move.w        #160,d0                ; Coordinata X
        move.w        #100,d1                ; Coordinata Y

        bsr.s        plotPIX                ; stampa il punto alla coord. X=d0, Y=d1

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
Finito:
        rts                        ; esci
```

```
*****************************************************************************
;                          Routine di plot dei punti (dots)
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)


LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

PlotPIX:
        move.w        d0,d2                ; Copia la coordinata X in d2


; Troviamo l'offset orizzontale, ossia la X

        lsr.w        #3,d0                ; Intanto trova l'offset orizzontale,
                                ; dividendo per 8 la coordinata X. Essendo lo
                                ; schermo fatto di bits, sappiamo che una
                                ; linea orizzontale e' larga 320 pixel, ossia
                                ; 320/8=40 bytes. Avendo la coordinata X che
                                ; va da 0 a 320, cioe' in bits, la dobbiamo
                                ; convertire in bytes, dividendola per 8.
                                ; In questo modo abbiamo il byte entro cui
                                ; settare il nostro bit.

; Ora troviamo l'offset verticale, ossia la Y:

        mulu.w        #largschermo,d1        ; moltiplica la larghezza di una linea per il
                                ; numero di linee, trovando l'offset
                                ; verticale dall'inizio dello schermo

; Infine troviamo l'offset dall'inizio dello schermo del byte dove si trova il
; punto (ossia il bit), che setteremo con l'istruzione BSET:

        add.w        d1,d0        ; Somma lo scostamento verticale a quello orizzontale

; Ora abbiamo in d0 l'offset, in bytes, dall'inizio dello schermo per trovare
; il byte dove si trova il punto da settare. Abbiamo quindi da scegliere quale
; degli 8 bit del byte va settato.

; Ora troviamo quale bit del byte dobbiamo settare:

        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X, ossia
                                ; l'offset (scostamento) nel byte,
                                ; ricavando in d2 il bit da settare
                                ; (in realta' sarebbe il resto della divisione
                                ;  per 8, fatta in precedenza)

        not.w        d2                ; Opportunamente nottato

; Ora abbiamo in d0 l'offset dall'inizio dello schermo per trovare il byte,
; in d2 il numero di bit da settare all'interno di quel bit, e in a0
; l'indirizzo del bitplane. Con una sola istruzione possiamo settare il bit:

        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                ; dall'inizio dello schermo.
```

```
        rts                     ; Esci.

*****************************************************************************

        SECTION         GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION         MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Ecco una semplice routine per stampare dei punti sullo schermo. Provate a cambiare le cordinate X e Y. Dovete ricordarvi che la coordinata 0,0 e' l'angolo in alto a sinistra, per cui 320,256 e' l'angolo in basso a destra. Il sistema di assi cartesiani ha la posizione 0,0 in basso a sinistra, per cui la Y e' invertita rispetto a tale riferimento. Se proprio si volesse fare in modo che 0,0 fosse in basso a destra basterebbero poche modifiche: partire dalla fine del bitplane e "tornare indietro" anziche' avanzare.

24.11 Lezione8m

```
; Lezione8m.s - Routine di stampa di punti (plot), usata in un loop per
;               fare una linea.

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************
```

```
; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                  -----a-bcdefghij

Coeff        equ        1        ; Coefficiente angolare, m

START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        lea        bitplane,a0        ; Indirizzo del bitplane dove stampare

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

;        Y=m*x, o meglio m*x=y, ossia Coeff*d0=d1

        Addq.W        #1,Miox                ; Incrementa la X
        move.w        Miox(PC),d1
        Mulu.w        #Coeff,d1        ; Y=m*x
        cmp.w        #255,d1                ; siamo in fondo allo schermo??
        bhi.s        Finito
        move.w        Miox(PC),d0        ; X

        bsr.s        plotPIX                ; stampa il punto alla coord. X=d0, Y=d1

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
Finito:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        Finito
        rts                        ; esci
```

```
MioX:
        dc.w        0

*****************************************************************************
;                       Routine di plot dei punti (dots)
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

PlotPIX:
        move.w        d0,d2               ; Copia la coordinata X in d2


; Troviamo l'offset orizzontale, ossia la X

        lsr.w        #3,d0                ; Intanto trova l'offset orizzontale,
                                  ; dividendo per 8 la coordinata X. Essendo lo
                                  ; schermo fatto di bits, sappiamo che una
                                  ; linea orizzontale e' larga 320 pixel, ossia
                                  ; 320/8=40 bytes. Avendo la coordinata X che
                                  ; va da 0 a 320, cioe' in bits, la dobbiamo
                                  ; convertire in bytes, dividendola per 8.
                                  ; In questo modo abbiamo il byte entro cui
                                  ; settare il nostro bit.

; Ora troviamo l'offset verticale, ossia la Y:

        mulu.w        #largschermo,d1        ; moltiplica la larghezza di una linea per il
                                  ; numero di linee, trovando l'offset
                                  ; verticale dall'inizio dello schermo

; Infine troviamo l'offset dall'inizio dello schermo del byte dove si trova il
; punto (ossia il bit), che setteremo con l'istruzione BSET:

        add.w        d1,d0        ; Somma lo scostamento verticale a quello orizzontale

; Ora abbiamo in d0 l'offset, in bytes, dall'inizio dello schermo per trovare
; il byte dove si trova il punto da settare. Abbiamo quindi da scegliere quale
; degli 8 bit del byte va settato.

; Ora troviamo quale bit del byte dobbiamo settare:

        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X, ossia
                                  ; l'offset (scostamento) nel byte,
                                  ; ricavando in d2 il bit da settare
                                  ; (in realta' sarebbe il resto della divisione
                                  ;  per 8, fatta in precedenza)

        not.w        d2                ; opportunamente nottato

; Ora abbiamo in d0 l'offset dall'inizio dello schermo per trovare il byte,
; in d2 il numero di bit da settare all'interno di quel bit, e in a0
; l'indirizzo del bitplane. Con una sola istruzione possiamo settare il bit:

        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
```

```
                              ; dall'inizio dello schermo.
        rts                      ; Esci.

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Con un loop facciamo una linea. Sappiamo che la "formula" di una linea e'
y=m*x, dove m e' un numero, detto coefficiente angolare, che determina tra
l'altro l'inclinazione della retta stessa. Per chi fosse azzerato con la
matematica, ecco in brevis cosa succede: se abbiamo che X e' 0, la Y la
ricaviamo moltiplicando la X per m, che qua e' definito con un EQU all'inizio
del listato. Supponiamo che Coeff sia = 1:

```
X = 0        -> Y=Coeff*X, ossia 1*0, ossia 0
```

Dunque, X=0 e Y=0

Nel loop incrementiamo di 1 la X. ecco cosa succede al prossimo loop:

```
X = 1        -> Y= 1*1, ossia 1
```

poi...

```
X = 2        -> Y=2
```

Insomma, la Y e' sempre uguale alla X, perche' la moltiplichiamo per Coeff=1.
Da cio' ne deriva la seguente linea:

```
11
  22
```

```
    33
      44
        55
          66
            77
              88
                ...

Chiaro perche' e' a 45 gradi??? Provate a cambiare il Coeff mettendo, ad
esempio, 2. In questo caso ecco cosa succede:

X = 0        -> Y=Coeff*X, ossia 2*0, ossia 0
X = 1        -> Y= 2*1, ossia 2
X = 2        -> Y= 2*2, ossia 4
X = 3        -> Y= 2*3, ossia 6
X = 4        -> Y= 2*4, ossia 8
X = 5        -> Y= 2*5, ossia 10

La linea che ne risulta, sara':

12

 24

  36

   48

    510


Ossia piu' verso sinistra, e non continua. ci sono dei "buchi" tra un punto e
l'altro perche' operiamo solo con i numeri interi, e i valori, per esempio,
tra 2 e 3 lasciano il "vuoto". Vedremo piu' avanti che ci sono molti modi per
usare numeri in "virgola mobile emulata" anche senza coprocessori matematici,
per esempio per fare calcoli in 3d.
```

Lezione8m2

```
; Lezione8m2.s - Routine di stampa di punti (plot), usata in un loop per
;                 fare delle linee.

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                -----a-bcdefghij


START:
```

```
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA         BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)               ; DMACON - abilita bitplane, copper
                                      ; e sprites.

        move.l        #COPPERLIST,$80(a5)         ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                 ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                 ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                 ; Disattiva l'AGA

        lea         bitplane,a0         ; Indirizzo del bitplane dove stampare

mouse:
        MOVE.L        #$1ff00,d1          ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2          ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0          ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                 ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                 ; aspetta la linea $130 (304)
        BNE.S         Waity1

        bsr.s         CalcolaLinea

        move.w         Miox(PC),d0         ; Coord. X
        move.w         Mioy(PC),d1         ; Coord  Y

        bsr.s         plotPIX                ; stampa il punto alla coord. X=d0, Y=d1

        MOVE.L         #$1ff00,d1          ; bit per la selezione tramite AND
        MOVE.L         #$13000,d2          ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L         4(A5),D0          ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L         D1,D0                 ; Seleziona solo i bit della pos. verticale
        CMPI.L         D2,D0                 ; aspetta la linea $130 (304)
        BEQ.S          Aspetta

        btst         #6,$bfe001          ; mouse premuto?
        bne.s         mouse
Finito:
        btst         #6,$bfe001          ; mouse premuto?
        bne.s         Finito
        rts                         ; esci

;        Y=m*x, o meglio m*x=y, ossia Coeff*d0=d1

CalcolaLinea:
        Addq.W         #1,Miox                ; Incrementa la X
        move.w         Miox(PC),d1
        Mulu.w         Coeff(PC),d1         ; y=m*x
        cmp.w         #255,d1                ; siamo in fondo allo schermo??
        blo.s         NonFinito
        addq.w         #1,Coeff          ; Aggiungi 1 al coefficiente angolare
        cmp.w         #8,Coeff          ; Abbiamo gia' fatto 7 linee?
        beq.s         Finito                 ; Se si, usciamo!
        clr.w         Miox                ; E riparti da 0 per la nuova linea con
```

```
                                ; diversa inclinazione
        moveq        #0,d1
NonFinito:
        move.w        d1,MioY
        rts

MioX:
        dc.w        0
MioY:
        dc.w        0

Coeff:
        dc.w        1

*****************************************************************************
;                        Routine di plot dei punti (dots)
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

PlotPIX:
        move.w        d0,d2                ; Copia la coordinata X in d2


; Troviamo l'offset orizzontale, ossia la X

        lsr.w        #3,d0                ; Intanto trova l'offset orizzontale,
                                ; dividendo per 8 la coordinata X. Essendo lo
                                ; schermo fatto di bits, sappiamo che una
                                ; linea orizzontale e' larga 320 pixel, ossia
                                ; 320/8=40 bytes. Avendo la coordinata X che
                                ; va da 0 a 320, cioe' in bits, la dobbiamo
                                ; convertire in bytes, dividendola per 8.
                                ; In questo modo abbiamo il byte entro cui
                                ; settare il nostro bit.

; Ora troviamo l'offset verticale, ossia la Y:

        mulu.w        #largschermo,d1        ; moltiplica la larghezza di una linea per il
                                ; numero di linee, trovando l'offset
                                ; verticale dall'inizio dello schermo

; Infine troviamo l'offset dall'inizio dello schermo del byte dove si trova il
; punto (ossia il bit), che setteremo con l'istruzione BSET:

        add.w        d1,d0        ; Somma lo scostamento verticale a quello orizzontale

; Ora abbiamo in d0 l'offset, in bytes, dall'inizio dello schermo per trovare
; il byte dove si trova il punto da settare. Abbiamo quindi da scegliere quale
; degli 8 bit del byte va settato.

; Ora troviamo quale bit del byte dobbiamo settare:

        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X, ossia
                                ; l'offset (scostamento) nel byte,
                                ; ricavando in d2 il bit da settare
```

```
                                ; (in realta' sarebbe il resto della divisione
                                ;  per 8, fatta in precedenza)

        not.w        d2                ; opportunamente nottato

; Ora abbiamo in d0 l'offset dall'inizio dello schermo per trovare il byte,
; in d2 il numero di bit da settare all'interno di quel bit, e in a0
; l'indirizzo del bitplane. Con una sola istruzione possiamo settare il bit:

        bset.b       d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                ; dall'inizio dello schermo.
        rts                        ; Esci.

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81         ; DiwStrt
        dc.w        $90,$2cc1         ; DiwStop
        dc.w        $92,$0038         ; DdfStart
        dc.w        $94,$00d0         ; DdfStop
        dc.w        $102,0                 ; BplCon1
        dc.w        $104,$24         ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                 ; Bpl1Mod
        dc.w        $10a,0                 ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000         ; color0 - SFONDO
        dc.w        $0182,$1af         ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE         ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256         ; un bitplane lowres 320x256

        end

In questo esempio il loop che fa una linea e' inserito in un altro loop per
fare varie linee cambiando il coefficiente angolare, che ora e' una word su
cui nessuno vieta di fare un "ADDQ".
```

Lezione8m3

```
; Lezione8m3.s - Routine di stampa di punti (plot), usata in un loop per
;                calcolare y=x*x, ossia una curva simile a quella prodotta
;                dalla caduta di un sasso in una frana (parabola!)

        Section        dotta,CODE
```

```
;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                 -----a-bcdefghij


START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        lea        bitplane,a0        ; Indirizzo del bitplane dove stampare

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

        bsr.s        CalcolaPotenza        ; y=x*x

        move.w        Miox(PC),d0        ; Coord. X
        move.w        Mioy(PC),d1        ; Coord  Y

        bsr.s        plotPIX                ; stampa il punto alla coord. X=d0, Y=d1

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
Finito:
```

```
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        Finito
        rts                         ; esci

;         _.--._     _
;        |   _ .|   (_)
;        |   \__|   ||
;        |______|   ||
;        .-‘--’-.   ||
;        | | |  |\__1|
;        |_| |__|__|_))
;         ||_| |    ||
;         |(_) |
;         |    |
;         |____|__
;         |______/g®m

;        Y=x*x, d0*d0=d1

CalcolaPotenza:
        Addq.W        #1,Miox               ; Incrementa la X
        move.w        Miox(PC),d1
        Mulu.w        d1,d1                ; y=m*x
        lsr.w        #3,d1                ; allarghiamo un po’ la parabbola
        cmp.w        #255,d1               ; siamo in fondo allo schermo??
        blo.s        NonFinito
        bra.s        Finito                ; Se si, usciamo!
        addq.w        #1,Coeff        ; Aggiungi 1 al coefficiente angolare
        cmp.w        #80,Coeff        ; Abbiamo gia’ fatto 39 linee?
NonFinito:
        move.w        d1,MioY
        rts

MioX:
        dc.w        0
MioY:
        dc.w        0
Coeff:
        dc.w        1

*****************************************************************************
;                        Routine di plot dei punti (dots)
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

PlotPIX:
        move.w        d0,d2                ; Copia la coordinata X in d2


; Troviamo l’offset orizzontale, ossia la X

        lsr.w        #3,d0                ; Intanto trova l’offset orizzontale,
                                ; dividendo per 8 la coordinata X. Essendo lo
                                ; schermo fatto di bits, sappiamo che una
                                ; linea orizzontale e’ larga 320 pixel, ossia
```

```
                                ; 320/8=40 bytes. Avendo la coordinata X che
                                ; va da 0 a 320, cioe' in bits, la dobbiamo
                                ; convertire in bytes, dividendola per 8.
                                ; In questo modo abbiamo il byte entro cui
                                ; settare il nostro bit.

; Ora troviamo l'offset verticale, ossia la Y:

        mulu.w        #largschermo,d1        ; moltiplica la larghezza di una linea per il
                                ; numero di linee, trovando l'offset
                                ; verticale dall'inizio dello schermo

; Infine troviamo l'offset dall'inizio dello schermo del byte dove si trova il
; punto (ossia il bit), che setteremo con l'istruzione BSET:

        add.w         d1,d0         ; Somma lo scostamento verticale a quello orizzontale

; Ora abbiamo in d0 l'offset, in bytes, dall'inizio dello schermo per trovare
; il byte dove si trova il punto da settare. Abbiamo quindi da scegliere quale
; degli 8 bit del byte va settato.

; Ora troviamo quale bit del byte dobbiamo settare:

        and.w         #%111,d2        ; Seleziona solo i primi 3 bit di X, ossia
                                ; l'offset (scostamento) nel byte,
                                ; ricavando in d2 il bit da settare
                                ; (in realta' sarebbe il resto della divisione
                                ;  per 8, fatta in precedenza)

        not.w         d2                ; opportunamente nottato

; Ora abbiamo in d0 l'offset dall'inizio dello schermo per trovare il byte,
; in d2 il numero di bit da settare all'interno di quel bit, e in a0
; l'indirizzo del bitplane. Con una sola istruzione possiamo settare il bit:

        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                ; dall'inizio dello schermo.
        rts                        ; Esci.

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE
```

```
        dc.w        $FFFF,$FFFE        ; Fine della copperlist

*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

Con questa variante si puo' ammirare una cosa del genere:

```
*
 *

  *


   *



    *




      *
```

Purtroppo lavoriamo solo con i numeri interi, e non sono disponibili i pixel
tra un punto e l'altro, che sono frazionari. Comunque meglio di niente!
Questo tipo di curva e' ottenuto assegnando il valore della y come la
elevazione al quadrato della x. In questo modo abbiamo:

```
X = 0        -> Y=X*X, ossia 0*0, ossia 0
X = 1        -> Y= 1*1, ossia 1
X = 2        -> Y= 2*2, ossia 4
X = 3        -> Y= 3*3, ossia 9
X = 4        -> Y= 4*4, ossia 16
X = 5        -> Y= 5*5, ossia 25
```

Esponenzialmente (exp) la Y aumenta rispetto alla X.

Lezione8m4

```
; Lezione8m4.s - Routine di stampa di punti (plot), usata in un loop per
;                 calcolare y=a*x*x, ossia delle parabole

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
```

```
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                  -----a-bcdefghij


START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA         BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                      ; e sprites.

        move.l        #COPPERLIST,$80(a5)         ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                 ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                 ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                 ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                 ; Disattiva l'AGA

        lea         bitplane,a0        ; Indirizzo del bitplane dove stampare

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

        bsr.s        CalcolaParabola        ; y=a*x*x

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
Finito:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        Finito
        rts                        ; esci

;        Y=a*x*x, coeff*d0*d0=d1

CalcolaParabola:
        Addq.W        #1,Miox                ; Incrementa la X
        move.w        Miox(PC),d1
```

```
        Mulu.w        d1,d1                ; x*x
        Mulu.w        Coeff(PC),d1        ; y=a*x*x
        lsr.w        #8,d1                ; dividi per 256 la Y per "allargare"

        cmp.w        #256,MioY        ; siamo sotto lo schermo??
        bhi.s        Riparti                ; se si, abbiamo 1 solo schermo!!! ripartiamo
        cmp.w        #319-160,MioX        ; siamo all'estrema destra dello schermo??
        ble.s        NonFinito
Riparti:
        addq.w        #1,Coeff        ; Aggiungi 1 al coefficiente della parabola
        cmp.w        #3,Coeff        ; Abbiamo gia' fatto 2 parabole?
        beq.s        Finito                ; Se si, usciamo!
        move.w        #-160,Miox        ; E riparti da X= -160 per la nuova parabola
        rts                        ; Niente da plottare questa volta.

NonFinito:
        move.w        d1,MioY

; Andiamo a plottare il punto:

        move.w        Miox(PC),d0        ; Coord. X
        add.w        #160,d0                ; spostati in avanti di 160, dato che calcolo
                                ; da -160 a +160, che devo normalizzare in
                                ; cordinate 0 fino 320... in questo modo ho
                                ; spostato la parabola a destra.
        move.w        Mioy(PC),d1        ; Coord  Y
        bsr.s        plotPIX                ; stampa il punto alla coord. X=d0, Y=d1

        rts


MioX:
        dc.w        -160        ; parto da -160 per "centrare" la parabola.
MioY:
        dc.w        0

Coeff:
        dc.w        1

*****************************************************************************
;                        Routine di plot dei punti (dots)
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

PlotPIX:
        move.w        d0,d2                ; Copia la coordinata X in d2
        lsr.w        #3,d0                ; Intanto trova l'offset orizzontale,
                                ; dividendo per 8 la coordinata X.
        mulu.w        #largschermo,d1
        add.w        d1,d0                ; Somma scost. verticale a quello orizzontale

        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X
                                ; (in realta' sarebbe il resto della divisione
                                ;  per 8, fatta in precedenza)
        not.w        d2
```

```
        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                  ; dall'inizio dello schermo.
        rts

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

La modifica piu' importante qua e' che abbiamo "spostato" la parabola verso
destra, "scoprendo" anche la parte negativa che sale:

```
         **
      *            *
   *              *
 *         0        *
       ZERO
```

Come si vede dallo schemuccio, con la X minore di zero la curva e' contraria,
o speculare, vah... Allora per vederla basta partire con una x di -160 e
arrivare fino a +160. Poi "spostiamo" il tutto a destra di 160, centrando la
parabola, con un semplice ADD.W #160,d0. -160 diventa 0 e +160 diventa 320.

In questo esempio abbiamo inserito anche un coefficiente per il quale
moltiplichiamo x*x, ottenendo di poter fare 2 parabole, una piu' "larga"
dell'altra.

Una nota finale: per rendere piu' "visibile" e meno tratteggiata la parabola,
viene divisa per 256 la coordinata Y con un LSR #8.

```
        lsr.w       #8,d1               ; dividi per 256 la Y per "allargare"

Come sapete, si puo' dividere o moltiplicare per potenze di 2 tramite lsr e
lsl, anche se non e' proprio "uguale" ad un MULU o DIVU. In questo caso
comunque funziona abbastanza...

P.S: Da questo listato in avanti la routine PlotPIX sara' senza i megacommenti
dei listati precedenti... e' inutile allungare cosi' i sorgente!
```

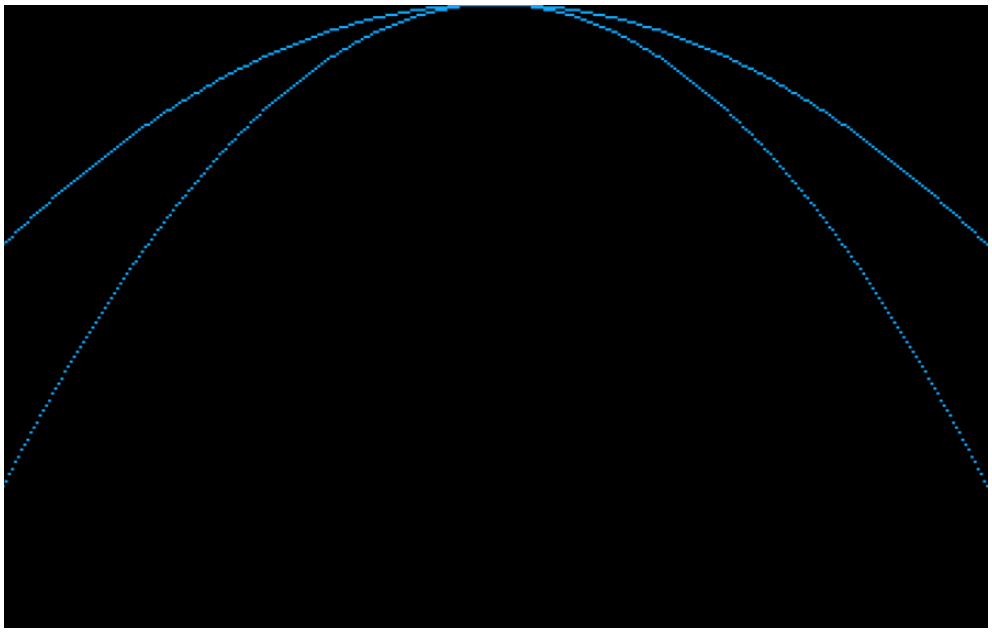


Figura 24.3: Lezione 8m4

Lezione8m5

```
; Lezione8m5.s - Routine di stampa di punti (plot), usata in un loop per
;                calcolare y=a*x*x, ossia delle parabole

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                   -----a-bcdefghij
```

```
START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA         BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap         d0
        move.w        d0,2(a1)

        MOVE.W        #DMASET,$96(a5)               ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)         ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        lea         bitplane,a0         ; Indirizzo del bitplane dove stampare

mouse:
        MOVE.L        #$1ff00,d1         ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2         ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0         ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S         Waity1

        bsr.s         CalcolaParabola         ; y=a*x*x

        MOVE.L        #$1ff00,d1         ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2         ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0         ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S         Aspetta

        btst.b        #6,$bfe001         ; mouse premuto?
        bne.s         mouse
Finito:
        btst.b        #6,$bfe001         ; mouse premuto?
        bne.s         Finito
        rts                         ; esci


;        Y=a*x*x, coeff*d0*d0=d1

CalcolaParabola:
        Addq.W        #1,Miox                ; Incrementa la X
        move.w        Miox(PC),d1
        Mulu.w        d1,d1                ; x*x
        Mulu.w        Coeff(PC),d1         ; y=a*x*x
        lsr.w         #8,d1                ; dividi per 256 la Y per "allargare"

        cmp.w         #255,MioY         ; siamo sotto lo schermo??
        bhi.s         Riparti                ; se si, abbiamo 1 solo schermo!!! ripartiamo
        cmp.w         #319-160,MioX         ; siamo all'estrema destra dello schermo??
        ble.s         NonFinito
```

```
Riparti:
        addq.w        #1,Coeff        ; Aggiungi 1 al coefficiente della parabola
        cmp.w        #6,Coeff        ; siamo gia' a Coeff=6
        beq.s        Finito                ; Se si, usciamo!
        tst.w        Coeff                ; Siamo allo zero?
        bne.s        OkCoeff                ; Se no, va bene
        addq.w        #1,Coeff        ; altrimenti saltiamo subito ad 1!
OkCoeff:
        move.w        #-160,Miox        ; E riparti da X= -160 per la nuova parabola
        rts                        ; Niente da plottare questa volta.

NonFinito:
        move.w        d1,MioY

; Andiamo a plottare il punto:

        move.w        Miox(PC),d0        ; Coord. X
        add.w        #160,d0                ; spostati in avanti di 160, dato che calcolo
                                ; da -160 a +160, che devo normalizzare in
                                ; cordinate 0 fino 320... in questo modo ho
                                ; spostato la parabola a destra.
        move.w        Mioy(PC),d1        ; Coord  Y
        bsr.s        plotPIX                ; stampa il punto alla coord. X=d0, Y=d1

        rts


MioX:
        dc.w        -160        ; parto da -160 per "centrare" la parabola.
MioY:
        dc.w        0

Coeff:
        dc.w        -5

*****************************************************************************
;                        Routine di plot dei punti (dots)
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

PlotPIX:
        move.w        d0,d2                ; Copia la coordinata X in d2
        lsr.w        #3,d0                ; Intanto trova l'offset orizzontale,
                                ; dividendo per 8 la coordinata X.
        mulu.w        #largschermo,d1
        add.w        d1,d0                ; Somma scost. verticale a quello orizzontale

        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X (resto)
        not.w        d2

        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                ; dall'inizio dello schermo.
        rts

*****************************************************************************
```

```
        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

        end
```

In questo listato l'unica modifica e' che usiamo anche coeff negativi.

24.12 Lezione8n

```
; Lezione8n.s - Routine di stampa di punti (plot), ottimizzata precalcolando
;                i multipli di 40 in una tabella, rimuovendo la moltiplicazione
;                nella routine PlotPix, che si prende il valore giusto dalla
;                tabella ogni volta.

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                  -----a-bcdefghij

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.
```

```
START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

; PRECALCOLIAMO UNA TABELLA CON I MULTIPLI DI 40, ossia della larghezza dello
; schermo, per evitare di fare una moltiplicazione per ogni plottaggio.

        lea        MulTab,a0        ; Indirizzo spazio di 256 words dove scrivere
                                ; i multipli di 40...
        moveq        #0,d0                ; Iniziamo da 0...
        move.w        #256-1,d7        ; Numero di multipli di 40 necessari
PreCalcLoop
        move.w        d0,(a0)+        ; Salviamo il multiplo attuale
        add.w        #LargSchermo,d0        ; aggiungiamo larghschermo, prossimo multiplo
        dbra        d7,PreCalcLoop        ; Creiamo tutta la MulTab

; Puntiamo la cop...

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

        move.w        #160,d0                ; coordinata X
        move.w        #128,d1                ; coordinata Y
        lea        bitplane,a0        ; Indirizzo del bitplane dove stampare in a0
        lea        MulTab,a1        ; Indirizzo della tabella con i multipli della
                                ; largh. schermo precalcolati in a1

        bsr.s        PlotPIXP        ; stampa il punto alla coord. X=d0, Y=d1

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
        rts                        ; esci
```

```
*****************************************************************************
;               Routine di plot dei punti (dots) ottimizzata
*****************************************************************************

;        Parametri in entrata di PlotPIXP:
;
;        a0 = Indirizzo bitplane destinazione
;        a1 = Indirizzo della tabella con i multipli di 40 precalcolati
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

PlotPIXP:
        move.w        d0,d2               ; Copia la coordinata X in d2
        lsr.w        #3,d0               ; Intanto trova l'offset orizzontale,
                                 ; dividendo per 8 la coordinata X.

; ** INIZIO MODIFICA: ecco le 2 istruzioni originali:
;
;        mulu.w        #largschermo,d1
;        add.w        d1,d0               ; Somma scost. verticale a quello orizzontale
;
; e quelle senza MULU:

; Ora troviamo l'offset verticale, ossia la Y, prendendo il giusto valore
; precalcolato dalla tabella Multab, il cui indirizzo e' in a1

        add.w        d1,d1               ; Moltiplichiamo la Y per 2, trovando l'offset
                                 ; dalla tabella dei multipli, infatti ogni
                                 ; multiplo e' una word, ossia 2 bytes. Ora, se
                                 ; per esempio la coordinata era 0, prendiamo
                                 ; il primo valore della tabella, che e' zero.
                                 ; Se e' 3, allora prendiamo il terzo valore
                                 ; della tabella, che pero' si trova al sesto
                                 ; byte, dato che dobbiamo saltare 2 bytes, 1
                                 ; word, per ogni valore in tabella.
        add.w        (a1,d1.w),d0        ; Aggiungiamo lo scostamento verticale giusto,
                                 ; preso dalla tabella, all'offset orizzontale

; ** FINE DELLA MODIFICA

        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X (resto)
        not.w        d2

        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                 ; dall'inizio dello schermo.
        rts

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
```

```
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

; Tabella che conterra' i multipli della larghezza dello schermo precalcolati
; per eliminare la moltiplicazione nella routine PlotPIX, aumentando la sua
; velocita'.

        SECTION        Precalc,bss

MulTab:
        ds.w        256

        end
```

Con questo listato facciamo una piccola introduzione alla lezione sulle ottimizzazioni, infatti "TABELLIAMO" una moltiplicazione. Questa operazione e' molto frequente nel codice delle piu' veloci demo o nei giochi 3d.
La nostra routine di stampa di un pixel funziona egregiamente, ma contiene una LENTISSIMA moltiplicazione. dobbiamo toglierla assolutamente. Non essendo una moltiplicazione per una potenza di 2, non possiamo sostituirlo con un LSL come abbiamo furbescamente fatto nella routine di print in Lezione8b.s.
Ma le vie del coding sono infinite. Considerate la situazione che abbiamo:

```
        mulu.w        #largschermo,d1
        add.w        d1,d0                ; Somma scost. verticale a quello orizzontale
```

Larghschermo in questo caso e' 40. In d1 abbiamo ogni volta un valore diverso, a seconda della Y, ma sappiamo che puo' andare da 0 a 255 come massimo.
Dunque ci sono 256 risultati possibili, a seconda che capiti uno dei 256 possibili valori di Y, ossia di d1. Questi 256 risultati, se dassimo ogni volta in entrata un numero crescende da 0 a 245 sarebbe:

0,40,80,120,160,200 ossia 40*0,40*1,40*2,40*3,40*4....

Immaginiamo di "prepararci" tutti questi 256 risultati possibili in uno spazio azzerato preventivamente preparato:

```
MulTab:
        ds.w        256
```

Per creare la tabella dei multipli di 40 basta un semplicissimo loop:

```
        lea        MulTab,a0        ; Indirizzo spazio di 256 words dove scrivere
                                ; i multipli di 40...
        moveq        #0,d0                ; Iniziamo da 0...
        move.w        #256-1,d7        ; Numero di multipli di 40 necessari
PreCalcLoop
```

```
        move.w        d0,(a0)+        ; Salviamo il multiplo attuale
        add.w         #LargSchermo,d0        ; aggiungiamo larghschermo, prossimo multiplo
        dbra         d7,PreCalcLoop        ; Creiamo tutta la MulTab
```

Ora abbiamo la tabella coi "risultati" pronti. Ma come facciamo a "prendere"
dalla tabella il risultato giusto ogni volta? In entrata abbiamo la coordinata
Y, ossia un numero da 0 a 255. Se Y e' zero, basta prendere il primo valore
della tabella, ossia la word $0000. Se invece fosse y=1, dobbiamo prendere il
secondo valore della tabella, che pero' si trova a 2 bytes dal suo inizio,
dato che i sui valori sono words. Allo stesso modo, se volessimo prendere il
risultato giusto per la coord. Y = 50, il risultato sarebbe la cinquantesima
word della tabella, ad una distanza cioe' di 100 bytes. tutto cio' non vi
suggerisce la soluzione? Per calcolare l'offset, ossia la distanza dall'inizio
della tabella, basta moltiplicare per 2 la Y! E siccome si puo' moltiplicare
per 2 con un:

```
         add.w        d1,d1
```

Siamo sempre senza moltiplicazioni. Ora in d1 abbiamo l'offset dall'inizio
della tabella; dobbiamo "prenderlo" e aggiungerlo a d0. Questo si puo' fare
con un'unica operazione:

```
        add.w        (a1,d1.w),d0        ; Aggiungiamo lo scostamento verticale giusto,
                                 ; preso dalla tabella, all'offset orizzontale
```

Avendo in a1 l'indirizzo della tabella MulTab.

Troveremo questo sistema di "tabellaggio" sempre piu' spesso nei listati
che svolgono molti calcoli.

Lezione8n2

```
; Lezione8n2.s - Routine di stampa di punti (plot), ottimizzata. Viene testata
;                la velocita' di questa routine al confronto con quella
;                non ottimizzata. Premere il tasto DESTRO del mouse per
;                far agire la routine ottimizzata, altrimenti agisce quella
;                normale.

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                 ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                -----a-bcdefghij

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
```

```
        LEA        BPLPOINTERS,A1
        move.w       d0,6(a1)
        swap        d0
        move.w       d0,2(a1)

; PRECALCOLIAMO UNA TABELLA CON I MULTIPLI DI 40, ossia della larghezza dello
; schermo, per evitare di fare una moltiplicazione per ogni plottaggio.

        lea        MulTab,a0        ; Indirizzo spazio di 256 words dove scrivere
                                    ; i multipli di 40...
        moveq        #0,d0                ; Iniziamo da 0...
        move.w       #256-1,d7        ; Numero di multipli di 40 necessari
PreCalcLoop
        move.w       d0,(a0)+         ; Salviamo il multiplo attuale
        add.w        #LargSchermo,d0        ; aggiungiamo larghschermo, prossimo multiplo
        dbra        d7,PreCalcLoop        ; Creiamo tutta la MulTab

; Puntiamo la cop...

        MOVE.W       #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l       #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w       d0,$88(a5)                ; Facciamo partire la COP
        move.w       #0,$1fc(a5)                ; Disattiva l'AGA
        move.w       #$c00,$106(a5)                ; Disattiva l'AGA
        move.w       #$11,$10c(a5)                ; Disattiva l'AGA

        lea        bitplane,a0        ; Indirizzo del bitplane dove stampare in a0
        lea        MulTab,a1        ; Indirizzo della tabella con i multipli della
                                    ; largh. schermo precalcolati in a1

mouse:
        bsr.s        Coordinate        ; loop delle coordinate per l'intero schermo
        move.w       MioX(PC),d0        ; coordinata X
        move.w       MioY(PC),d1        ; coordinata Y

        btst        #2,$16(a5)        ; tsato destro del mouse premuto?
        beq.s        Ottimizzata
        btst.b       #1,FaiSfai        ; Azzerare o Settare?
        bne.s        Sfai
        bsr.s        PlotPIX                ; stampa il punto alla coord. X=d0, Y=d1
        bra.s        OkPlottato
Sfai:
        bsr.s        ErasePIX        ; azzera il punto alla coord. X=d0, Y=d1
        bra.s        OkPlottato

Ottimizzata:
        btst.b       #1,FaiSfai        ; Azzerare o Settare?
        bne.s        SfaiP
        bsr.w        PlotPIXP        ; stampa il punto alla coord. X=d0, Y=d1
        bra.s        OkPlottato
SfaiP:
        bsr.w        ErasePIXP        ; azzera il punto alla coord. X=d0, Y=d1
OkPlottato:
        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
        rts                        ; esci



MioX:
```

```
        dc.w        0
MioY:
        dc.w        0
FaiSfai:
        dc.w        0

; Routinetta che fa stampare e cancellare continuamente tutti lo schermo un
; punto per volta.

Coordinate:
        addq.w       #1,MioX               ; prossimo pixel sulla linea
        cmp.w       #320,MioX        ; ultimo pixel di questa linea?
        beq.s       FinitoLinea         ; se si, cominciamo quella sotto!
        rts                         ; altrimenti, facciamo questo punto!

FinitoLinea:
        clr.w       MioX                ; ripartiamo dall'inizio della riga
        addq.w       #1,MioY                ; alla riga sotto...
        cmp.w       #256,MioY        ; Abbiamo finito la schermata? Ultima riga?
        beq.s       Cambiariparti
        rts

CambiaRiparti:
        bchg.b       #1,FaiSfai        ; Cambia lo stato di scrittura/cancellazione
        clr.w       MioX                ; e riparti da coordinata X=0
        clr.w       MioY                ; Y=0
        rts

*****************************************************************************
;               Routine di plot dei punti (dots) normale
*****************************************************************************

;        Parametri in entrata di PlotPIX:
;
;        a0 = Indirizzo bitplane destinazione
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)


PlotPIX:
        move.w       d0,d2                ; Copia la coordinata X in d2
        lsr.w       #3,d0                ; Intanto trova l'offset orizzontale,
                            ; dividendo per 8 la coordinata X.
        mulu.w       #largschermo,d1
        add.w       d1,d0                ; Somma scost. verticale a quello orizzontale

        and.w       #%111,d2        ; Seleziona solo i primi 3 bit di X (resto)
        not.w       d2

        bset.b       d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                            ; dall'inizio dello schermo.
        rts

; Routine che CANCELLA un pixel. Basta sostituire BCLR a BSET.

ErasePIX:
        move.w       d0,d2                ; Copia la coordinata X in d2
        lsr.w       #3,d0                ; Intanto trova l'offset orizzontale,
                            ; dividendo per 8 la coordinata X.
        mulu.w       #largschermo,d1
        add.w       d1,d0                ; Somma scost. verticale a quello orizzontale
```

```
        and.w       #%111,d2        ; Seleziona solo i primi 3 bit di X (resto)
        not.w       d2

        bclr.b       d2,(a0,d0.w)        ; Azzera il bit d2 del byte distante d0 bytes
                            ; dall'inizio dello schermo.
        rts

*****************************************************************************
;               Routine di plot dei punti (dots) ottimizzata
*****************************************************************************

;        Parametri in entrata di PlotPIXP:
;
;        a0 = Indirizzo bitplane destinazione
;        a1 = Indirizzo della tabella con i multipli di 40 precalcolati
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

PlotPIXP:
        move.w       d0,d2               ; Copia la coordinata X in d2
        lsr.w       #3,d0               ; Intanto trova l'offset orizzontale,
                            ; dividendo per 8 la coordinata X.
        add.w       d1,d1               ; Moltiplichiamo la Y per 2, trovando l'offset
        add.w       (a1,d1.w),d0        ; scostamento verticale + offset orizzontale
        and.w       #%111,d2        ; Seleziona solo i primi 3 bit di X
        not.w       d2                ; nottati
        bset        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                            ; dall'inizio dello schermo.
        rts

; Routine che CANCELLA un pixel. Basta sostituire BCLR a BSET.

ErasePIXP:
        move.w       d0,d2               ; Copia la coordinata X in d2
        lsr.w       #3,d0               ; Intanto trova l'offset orizzontale,
                            ; dividendo per 8 la coordinata X.
        add.w       d1,d1               ; Moltiplichiamo la Y per 2, trovando l'offset
        add.w       (a1,d1.w),d0        ; scostamento verticale + offset orizzontale
        and.w       #%111,d2        ; Seleziona solo i primi 3 bit di X
        not.w       d2                ; nottati
        bclr        d2,(a0,d0.w)        ; azzera il bit d2 del byte distante d0 bytes
                            ; dall'inizio dello schermo.
        rts

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
```

```
        dc.w $e0,0,$e2,0        ;primo          bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C

BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

; Tabella che conterra' i multipli della larghezza dello schermo precalcolati
; per eliminare la moltiplicazione nella routine PlotPIX, aumentando la sua
; velocita'.

        SECTION        Precalc,bss

MulTab:
        ds.w        256

        end
```

Questo listato vuole essere un test per verificare se veramente la routine senza moltiplicazione va piu' veloce. A questo scopo viene disegnato tutto lo schermo e ricancellato con degli "ErasePIX" che non sono altro che la routine normale con BCLR al posto di BSET. Normalmente viene eseguita la routine non ottimizzata, tenendo premuto il tasto destro si esegue quella ottimizzata. A seconda dei computer e della presenza della fast ram la fisserenza sara' diversa. Per esempio se la tabella va in CHIP ram anziche' in FAST RAM la valocita' acquistata e' minore. Per esempio, sul 68040 le moltiplicazioni sono molto velocizzate rispetto ai precedenti processori, al punto che se si esegue questo listato senza fastram e con le caches disattivate e' piu' lenta la routine senza moltiplicazione, dato che deve accedere alla tabella in CHIP RAM. Comunque chi ha A4000 ha anche la fast ram, tranquilli, inoltre su 68030 o inferiori togliere una moltiplicazione e' sempre una buona azione.

Lezione8n3

```
; Lezione8n3.s - Routine di stampa di punti (plot), ottimizzata.
;                Viene usata una tabella per "muovere" un punto. Tasto destro
;                per far "lasciare la scia" al punto.

        Section        dotta,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
```

Figura 24.4: Lezione 8n2

```
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                  -----a-bcdefghij

LargSchermo        equ        40        ; Larghezza dello schermo in bytes.

START:
;         PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L        #BITPLANE,d0
        LEA        BPLPOINTERS,A1
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)

; PRECALCOLIAMO UNA TABELLA CON I MULTIPLI DI 40, ossia della larghezza dello
; schermo, per evitare di fare una moltiplicazione per ogni plottaggio.

        lea        MulTab,a0        ; Indirizzo spazio di 256 words dove scrivere
                                ; i multipli di 40...
        moveq        #0,d0                ; Iniziamo da 0...
        move.w        #256-1,d7        ; Numero di multipli di 40 necessari
PreCalcLoop
        move.w        d0,(a0)+        ; Salviamo il multiplo attuale
        add.w        #LargSchermo,d0        ; aggiungiamo larghschermo, prossimo multiplo
        dbra        d7,PreCalcLoop        ; Creiamo tutta la MulTab

; Puntiamo la cop...

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
```

```
        move.w        #$11,$10c(a5)               ; Disattiva l'AGA

        lea        bitplane,a0        ; Indirizzo del bitplane dove stampare in a0
        lea        MulTab,a1        ; Indirizzo della tabella con i multipli della
                              ; largh. schermo precalcolati in a1

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BNE.S        Waity1

        bsr.w        LeggiTabelle        ; Legge le posizioni X ed Y dalle tabelle

        move.w        MioX(PC),d0        ; coordinata X
        move.w        MioY(PC),d1        ; coordinata Y

        bsr.w        PlotPIXP        ; stampa il punto alla coord. X=d0, Y=d1

        btst        #2,$16(a5)        ; tasto destro del mouse premuto?
        beq.s        NonCancellare

        move.w        MioXold(PC),d0        ; coordinata X vecchia da cancellare
        move.w        MioYold(PC),d1        ; coordinata Y vecchia

        bsr.w        ErasePIXP        ; azzera il punto alla coord. X=d0, Y=d1

NonCancellare:
        move.w        MioX(PC),MioXold ; prepara le coord del punto che cancelleremo
        move.w        MioY(PC),MioYold ; dopo

        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$13000,d2        ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $130 (304)
        BEQ.S        Aspetta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse
        rts                        ; esci



MioX:
        dc.w        0
MioY:
        dc.w        0
MioXold:
        dc.w        0
MioYold:
        dc.w        0


*****************************************************************************
;                Routine di plot dei punti (dots) ottimizzata
*****************************************************************************
```

```
;        Parametri in entrata di PlotPIXP:
;
;        a0 = Indirizzo bitplane destinazione
;        a1 = Indirizzo della tabella con i multipli di 40 precalcolati
;        d0.w = Coordinata X (0-319)
;        d1.w = Coordinata Y (0-255)

;            .....
;          __\ oO/__
;         / _ \./ _ \
;        /\/|  "  |\/\
;        \ \|_____|/ /
;         \ \_(_)_| /
;          \\\     \
;         /   \/    \
;         \____\____/
;        (_____\_____)eD
;

PlotPIXP:
        move.w        d0,d2               ; Copia la coordinata X in d2
        lsr.w        #3,d0               ; Intanto trova l'offset orizzontale,
                                 ; dividendo per 8 la coordinata X.
        add.w        d1,d1               ; Moltiplichiamo la Y per 2, trovando l'offset
        add.w        (a1,d1.w),d0        ; scostamento verticale + offset orizzontale
        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X (resto)
        not.w        d2                ; nottati
        bset.b        d2,(a0,d0.w)        ; Setta il bit d2 del byte distante d0 bytes
                                 ; dall'inizio dello schermo.
        rts

; Routine che CANCELLA un pixel. Basta sostituire BCLR a BSET.

ErasePIXP:
        move.w        d0,d2               ; Copia la coordinata X in d2
        lsr.w        #3,d0               ; Intanto trova l'offset orizzontale,
                                 ; dividendo per 8 la coordinata X.
        add.w        d1,d1               ; Moltiplichiamo la Y per 2, trovando l'offset
        add.w        (a1,d1.w),d0        ; scostamento verticale + offset orizzontale
        and.w        #%111,d2        ; Seleziona solo i primi 3 bit di X (resto)
        not.w        d2                ; nottati
        bclr.b        d2,(a0,d0.w)        ; azzera il bit d2 del byte distante d0 bytes
                                 ; dall'inizio dello schermo.
        rts

*****************************************************************************

LeggiTabelle:
        move.l        a0,-(SP)        ; salva a0 nello stack
        ADDQ.L        #1,TABYPOINT         ; Fai puntare al byte successivo
        MOVE.L        TABYPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                  ; copiato in a0
        CMP.L        #FINETABY-1,A0  ; Siamo all'ultimo byte della TAB?
        BNE.S        NOBSTARTY        ; non ancora? allora continua
        MOVE.L        #TABY-1,TABYPOINT ; Riparti a puntare dal primo byte
NOBSTARTY:
        moveq        #0,d0               ; Pulisci d0
        MOVE.b        (A0),d0                ; copia il byte della tabella, cioe' la
                                 ; coordinata Y in d0 in modo da farla
                                 ; trovare alla routine universale

        ADDQ.L        #2,TABXPOINT         ; Fai puntare alla word successiva
```

```
        MOVE.L        TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                ; copiato in a0
        CMP.L        #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S        NOBSTARTX        ; non ancora? allora continua
        MOVE.L        #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq        #0,d1                ; azzeriamo d1
        MOVE.w        (A0),d1                ; poniamo il valore della tabella, cioe'
                                ; la coordinata X in d1
        move.w        d0,MioY                ; salva le coordinate
        move.w        d1,MioX
        move.l        (sp)+,a0        ; riprendi a0 dallo stack
        rts


TABYPOINT:
        dc.l        TABY-1                ; NOTA: i valori della tabella qua sono bytes
TABXPOINT:
        dc.l        TABX-2                ; NOTA: i valori della tabella qua sono word

; Tabella con coordinate Y

TABY:
        incbin        "ycoordinatok.tab"        ; 200 valori .B
FINETABY:

; Tabella con coordinate X

TABX:
        incbin        "xcoordinatok.tab"        ; 150 valori .W
FINETABX:

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:

        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w        $108,0                ; Bpl1Mod
        dc.w        $10a,0                ; Bpl2Mod
                    ; 5432109876543210
        dc.w        $100,%0001001000000000        ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo         bitplane

        dc.w        $0180,$000        ; color0 - SFONDO
        dc.w        $0182,$1af        ; color1 - SCRITTE

        dc.w        $FFFF,$FFFE        ; Fine della copperlist


*****************************************************************************

        SECTION        MIOPLANE,BSS_C
```

```
BITPLANE:
        ds.b        40*256        ; un bitplane lowres 320x256

; Tabella che conterra' i multipli della larghezza dello schermo precalcolati
; per eliminare la moltiplicazione nella routine PlotPIX, aumentando la sua
; velocita'.

        SECTION        Precalc,bss

MulTab:
        ds.w        256

        end
```

In questo esempio abbiamo semplicemente aggiunto la routine che legge dalle 2 tabelle le cordinate X ed Y, come per gli sprite. Come si puo' vedere e' utile anche per routines di punti. Facendo tabelle e routines piu' complesse si possono ottenere onde varie.

Lezione8n4

```
; Lezione8n4.s - Routine di stampa di punti (plot), su diversi planes,
;                usata per stampare un disegno (le coordinate sono lette
;                da una tabella che potete scrivervi a mano o in altri modi)
;
; Autore:        Lorenzo Di Gaetano        ( The Amiga Dj )
;

        SECTION        Powerplot,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; con questo include mi risparmio di
                                ; riscriverla ogni volta!
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; copper e bitplane DMA abilitati
;                -----a-bcdefghij


START:
        move.l        #Bitplanes,d0
        lea        Bplpointers,a1
        moveq        #3-1,d1
Pointbp:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        add.l        #40*256,d0
        add.w        #8,a1
        dbra        d1,pointbp

        lea        $dff000,a5
        move.w        #DMASET,$96(a5)
```

```
        move.l        #coplist,$80(a5)        ; punta la cop
        move.l        d0,$88(a5)

        move.w        #0,$1fc(a5)               ; Disattiva l'AGA
        move.w        #$c00,$106(a5)               ; Disattiva l'AGA
        move.w        #$11,$10c(a5)               ; Disattiva l'AGA

; STAMPA DEL "CIAO"

        move.w        #(256-1),d7        ; (Numero Dati/2)-1        -> numero punti!
Loop:
        bsr.w         WaitBlank        ; Aspetta la linea 255 per sincronizzare
        move.l         POINT(PC),a2        ; coordinata attuale in tab dal puntatore
        addq.l         #2,POINT        ; Sposta il puntatore alle prossime coord.
                                ; per il prossimo loop. (prossimo punto!)
        moveq         #0,d0
        moveq         #0,d1
        move.b         (a2)+,d0        ; coord. X
        move.b         (a2)+,d1        ; coord. Y
        moveq         #6,d2                ; colore = 6 (bianco)
        lea         Bitplanes,a0        ; Planes di Destinazione
        bsr.w         Plot                ; Stampa il punto
                                ; a0 = Indirizzo bitplanes
                                ; d0 = Coordinata X
                                ; d1 = Coordinata Y
                                ; d2 = Colore del pixel
        dbra         d7,loop                ; esegui d7 volte, quindi stampa d7 punti!

; STAMPA DI "AMIGA"

        move.w        #(53-1),d7        ; (Numero Dati/2)-1
Loop2:
        bsr.w         WaitBlank        ; Aspetta la linea 255 per sincronizzare
        move.l         POINT(PC),a2        ; coordinata attuale in tab dal puntatore
        addq.l         #2,POINT        ; Sposta il puntatore alle prossime coord.
                                ; per il prossimo loop. (prossimo punto!)
        moveq         #0,d0
        moveq         #0,d1
        move.b         (a2)+,d0        ; coord. X
        move.b         (a2)+,d1        ; coord. Y
        moveq         #2,d2                ; colore = 2 (verde)
        lea         Bitplanes,a0        ; Planes di Destinazione
        bsr.w         Plot                ; Stampa il punto
        dbra         d7,loop2        ; esegui d7 volte, quindi stampa d7 punti!

; STAMPA DI "LIVES"

        move.w        #(45-1),d7        ; (Numero Dati/2)-1
Loop3:
        bsr.w         WaitBlank        ; Aspetta la linea 255 per sincronizzare
        move.l         POINT(PC),a2        ; coordinata attuale in tab dal puntatore
        addq.l         #2,POINT        ; Sposta il puntatore alle prossime coord.
                                ; per il prossimo loop. (prossimo punto!)
        moveq         #0,d0
        moveq         #0,d1
        move.b         (a2)+,d0        ; coord. X
        move.b         (a2)+,d1        ; coord. Y
        moveq         #4,d2                ; colore = 4 (giallo)
        lea         Bitplanes,a0        ; Planes di Destinazione
        bsr.w         Plot                ; Stampa il punto
        dbra         d7,loop3        ; esegui d7 volte, quindi stampa d7 punti!
```

```
; STAMPA DI "T"

        move.w        #(56-1),d7        ; (Numero Dati/2)-1
Loop4:
        bsr.s         WaitBlank         ; Aspetta la linea 255 per sincronizzare
        move.l         POINT(PC),a2        ; coordinata attuale in tab dal puntatore
        addq.l         #2,POINT         ; Sposta il puntatore alle prossime coord.
                                  ; per il prossimo loop. (prossimo punto!)
        moveq         #0,d0
        moveq         #0,d1
        move.b         (a2)+,d0         ; coord. X
        move.b         (a2)+,d1         ; coord. Y
        moveq         #1,d2                ; colore = 1 (Rosso)
        lea         Bitplanes,a0         ; Planes di Destinazione
        bsr.w         Plot                 ; Stampa il punto
        dbra         d7,loop4         ; esegui d7 volte, quindi stampa d7 punti!

; STAMPA DI "A"

        move.w        #(45-1),d7        ; (Numero Dati/2)-1
Loop5:
        bsr.s         WaitBlank         ; Aspetta la linea 255 per sincronizzare
        move.l         POINT(PC),a2        ; coordinata attuale in tab dal puntatore
        addq.l         #2,POINT         ; Sposta il puntatore alle prossime coord.
                                  ; per il prossimo loop. (prossimo punto!)
        moveq         #0,d0
        moveq         #0,d1
        move.b         (a2)+,d0         ; coord. X
        move.b         (a2)+,d1         ; coord. Y
        moveq         #7,d2                ; colore = 7 (Viola)
        lea         Bitplanes,a0         ; Planes di Destinazione
        bsr.s         Plot                 ; Stampa il punto
        dbra         d7,loop5         ; esegui d7 volte, quindi stampa d7 punti!

; STAMPA DI "D"

        move.w        #(53-1),d7        ; (Numero Dati/2)-1
Loop6:
        bsr.s         WaitBlank         ; Aspetta la linea 255 per sincronizzare
        move.l         POINT(PC),a2        ; coordinata attuale in tab dal puntatore
        addq.l         #2,POINT         ; Sposta il puntatore alle prossime coord.
                                  ; per il prossimo loop. (prossimo punto!)
        moveq         #0,d0
        moveq         #0,d1
        move.b         (a2)+,d0         ; coord. X
        move.b         (a2)+,d1         ; coord. Y
        moveq         #5,d2                ; colore = 5 (azzurro)
        lea         Bitplanes,a0         ; Planes di Destinazione
        bsr.s         Plot                 ; Stampa il punto
        dbra         d7,loop6         ; esegui d7 volte, quindi stampa d7 punti!


mouse:
        btst         #6,$bfe001         ; tasto sinistro del mouse premuto?
        bne.s         mouse
        rts

*****************************************************************************
;                Routine che attende la linea 255
*****************************************************************************

WaitBlank:
```

```
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$12c00,d2        ; linea da aspettare = $12c
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0               ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0               ; aspetta la linea $12c
        BNE.S        Waity1
Aspetta:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0               ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0               ; aspetta la linea $12c+1
        BEQ.S        Aspetta
        RTS

*****************************************************************************
;---------------------------------------------------------------------------
; Routine di plotpix con gestione del colore, fino a 8 bitplane (aga)
; by Lorenzo "TAD" Di Gaetano 1995
;
; Parametri di entrata:
;
; a0 = Indirizzo bitplanes
; d0.w = Coordinata X
; d1.w = Coordinata Y
; d2.b = Colore del pixel
;---------------------------------------------------------------------------
*****************************************************************************

;            .....
;          __\ oO/__
;         / _ \./ _ \
;        /\/|  "  |\/\
;        \ \|_____|/ /
;         \ \_(_)_| /
;          \\\     \
;         /   \/    \
;         \____\____/
;        (_____\_____)eD
;

Plot:
        movem.l         d0-d7/a0-a6,-(SP)
        mulu.w         #40,d1                          ; Moltiplichiamo per 40 la Y
                                                ; ottenendo cosi‘ l‘ offset verticale
        move.w         d0,d3                           ; Salviamo la X in d3
        lsr.w         #3,d0                           ; Dividiamo per 8 la x in modo da
                                                ; trovare l‘ offset orizzontale.
        add.w         d0,d1                           ; Sommiamo offset verticale e
                                                ; orizzontale in modo da trovare
                                                ; l‘ offset globale.
        move.l         d1,a1                           ; Salviamo d1 in a1
        add.l         a0,a1                           ; Sommiamo l‘ offset all‘ indirizzo
                                                ; dei bitplane trovando il byte
                                                ; giusto su cui operare.

        and.l         #%0000000000001111,d3         ; Selezioniamo i 3 bit bassi per
                                                ; trovare il bit giusto da settare
        not.w         d3                              ; Lo invertiamo

; A questo punto abbiamo:
;
; in a1 l’indirizzo del byte dove dovremo fare il bset (al primo plane)
```

```
; in d3 il bit da settare
; in d2 il numero del colore

; Ora dobbiamo settare i bit dei plani giusti per avere il colore in d2!!

        moveq        #3-1,d4                      ; Numero possibili bitplane...
        moveq        #0,d5                       ; d5 (contatore bit) azzerato
Colorloop:
        move.l        a1,a2                       ; Salviamo a1 (plane1) in a2
        move.l        d5,d6                       ; e d5 (contatore bit) in d6, per
                                        ; calcolare l'eventuale offset
                                        ; dal primo bitplane.
        btst.l        d5,d2                       ; E' qui il trucco. Testiamo i bit
                                        ; che compongono il valore del colore
                                        ; in modo da settare i corrispondenti
                                        ; bit dei vari bitplane!
        beq.s         Salta                       ; Bit azzerato, allora si salta.
                                        ; Se settato invece continua,
                                        ; settando il bit nel plane giusto!
        mulu.w        #40*256,d6              ; In d6 abbiamo il numero del
                                        ; bitplane in esame, lo moltiplichiamo
                                        ; per la grandezza del bitplane
                                        ; trovandone l' esatto indirizzo.
        add.l         d6,a2                       ; Aggiungiamo l' offset al plane 1
        bset.b        d3,(a2)                       ; E finalmente disegnamo il pixel
                                        ; nel plane giusto.
Salta:
        addq.b        #1,d5                       ; Prossimo bit da testare per trovare
                                        ; in quali plane fare il bset!
        dbra         d4,Colorloop              ; Ripeti loop
        movem.l        (SP)+,d0-d7/a0-a6
        rts

; Puntatore alla tabella

POINT:
        dc.l         TAB

; Tabella con le coordinate, nel formato: X,Y punto1, X,Y, punto2, X,Y....
; da notare che le cordinate sono in .bytes, quindi da 0 a 255, per cui
; la coordinata X non puo' andare fino a 320... se si volesse raggiungere
; una coordinata X maggiore di 255 si potrebbe fare la tabella in words.

TAB:
 dc.b        130,106,129,106,128,106,127,106,126,106,125,107,124,107,124,108
 dc.b        123,108,122,109,121,109,121,110,120,110,120,111,120,112,119,113
 dc.b        119,114,119,115,118,115,118,116,118,117,118,118,117,118,117,119
 dc.b        117,120,116,121,116,122,116,123,116,124,116,125,115,126,115,127
 dc.b        115,128,115,129,115,130,115,131,115,132,115,133,115,134,115,135
 dc.b        116,135,116,136,116,137,116,138,117,138,117,139,117,140,118,140
 dc.b        119,140,119,141,120,141,120,142,121,142,122,142,122,143,123,143
 dc.b        124,143,125,143,126,143,127,143,128,143,129,143,130,143,131,143
 dc.b        131,142,132,142,132,141,133,141,133,140,134,140,134,139,134,138
 dc.b        134,137,135,136,135,135,135,134,135,133,135,132,135,131,135,130
 dc.b        135,129,135,120,136,130,136,131,136,132,136,133,136,134,136,135
 dc.b        136,136,136,137,136,138,136,139,136,140,137,141,137,142,138,142
 dc.b        138,143,139,143,140,143,140,144,141,144,142,144,143,144,144,144
 dc.b        144,143,145,143,145,142,146,141,146,140,146,139,147,138,147,137
 dc.b        148,136,148,135,149,135,149,134,149,133,149,132,149,131,149,130
 dc.b        149,129,149,128,149,127,150,127,150,126,151,126,151,125,152,125
 dc.b        153,125,154,125,155,125,156,125,157,125,157,126,158,126,159,126
 dc.b        160,127,160,128,161,128,162,129,162,130,162,131,162,132,162,133
```

```
 dc.b        162,134,162,135,162,136,162,137,161,138,161,139,160,140,160,141
 dc.b        159,142,159,143,158,143,157,143,156,143,155,143,154,143,154,142
 dc.b        153,142,152,142,152,141,151,141,150,141,150,140,150,139,150,138
 dc.b        149,138,149,137,149,136,163,129,163,130,163,131,163,132,163,133
 dc.b        163,134,163,135,163,136,163,137,163,138,163,139,163,140,164,141
 dc.b        164,142,165,142,166,142,167,142,168,142,169,142,169,141,170,141
 dc.b        170,140,171,140,171,139,172,139,173,138,173,137,174,137,174,136
 dc.b        174,135,174,134,174,133,174,132,174,131,175,130,175,129,175,128
 dc.b        176,128,176,127,176,126,177,125,177,124,178,124,178,123,179,122
 dc.b        180,122,181,122,182,122,183,122,184,122,185,122,186,123,187,124
 dc.b        188,125,188,126,188,127,188,128,188,129,188,130,188,131,188,132
 dc.b        188,133,188,134,188,135,188,136,187,137,187,138,186,138,186,139
 dc.b        185,140,185,141,184,141,183,142,182,142,181,142,180,142,179,142
 dc.b        178,142,177,142,176,141,175,141,175,140,174,140,174,139,174,138
 dc.b        107,155,107,154,107,153,107,152,108,151,109,151,110,152,110,153
 dc.b        110,154,110,155,108,154,109,154,112,155,112,154,112,153,112,152
 dc.b        112,151,115,151,115,152,115,153,115,154,115,155,113,152,114,152
 dc.b        117,151,117,152,117,153,117,154,117,155,122,151,121,151,120,151
 dc.b        119,152,119,153,119,154,120,155,121,155,122,155,122,154,122,153
 dc.b        121,153,124,155,124,154,124,153,124,152,125,151,126,151,127,152
 dc.b        127,153,127,154,127,155,125,154,126,154,131,151,131,152,131,153
 dc.b        131,154,131,155,132,155,133,155,135,151,135,152,135,153,135,154
 dc.b        135,155,137,151,137,152,137,153,137,154,138,155,139,154,139,153
 dc.b        139,152,139,151,141,151,141,152,141,153,141,154,141,155,142,151
 dc.b        143,151,142,153,142,155,143,155,148,151,147,151,146,151,145,152
 dc.b        146,153,147,153,148,154,147,155,146,155,145,155,151,151,151,152
 dc.b        151,153,151,155
 dc.b        163,155,162,155,161,156,160,156,159,157,159,156,159,155,159,154
 dc.b        159,153,159,152,159,151,160,151,161,151,162,151,163,151,164,151
 dc.b        165,151,166,151,167,151,168,151,169,151,170,151,171,151
 dc.b        171,152,171,153
 dc.b        171,154,171,155,171,156,171,157,170,156,169,156,168,155,167,155
 dc.b        167,156,167,157,168,158,168,159,168,160,168,161,169,162,169,163
 dc.b        168,163,167,163,166,163,165,163,164,163,163,163,162,163,161,163
 dc.b        161,162,162,161,162,160,162,159,162,158,163,157,163,156,176,151
 dc.b        177,151,178,151,178,152,179,153,179,154,179,155,180,156,180,157
 dc.b        180,158,181,159,181,160,181,161,182,162,182,163,181,163,180,163
 dc.b        179,163,179,162,179,161,178,161,177,161,176,161,175,161,175,162
 dc.b        175,163,174,163,173,163,172,163,172,162,173,161,173,160,173,159
 dc.b        174,158,174,157,174,156,175,155,175,154,175,153,176,152,177,156
 dc.b        177,157,178,158,177,158,176,158,185,151,186,151,187,151,188,151
 dc.b        189,151,190,151,191,151,192,151,193,152,194,153,194,154,194,155
 dc.b        194,156,194,157,194,158,194,159,194,160,194,161,193,162,192,163
 dc.b        191,163,190,163,189,163,188,163,187,163,186,163,185,162,185,161
 dc.b        185,160,186,160,186,159,186,158,186,157,186,156,186,155,186,154
 dc.b        185,154,185,153,185,152,189,154,190,154,191,155,191,156,191,157
 dc.b        191,158,191,159,190,160,189,160,189,159,189,158,189,157,189,156
 dc.b        189,155

;*****************************************************************************
;                               Copperlist
;*****************************************************************************

        SECTION         Powercopper,DATA_C

Coplist:
        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
```

```
        dc.w        $108,0                  ; Bpl1Mod
        dc.w        $10a,0                  ; Bpl2Mod

bplpointers:
        dc.w        $e0,0,$e2,0         ; plane1
        dc.w        $e4,0,$e6,0         ; plane2
        dc.w        $e8,0,$ea,0         ; plane3

                    ; 5432109876543210
        dc.w        $100,%0011001000000000        ; 3 bitplane LOWRES 320x256

        dc.w        $180,$0000          ; color0 - Nero
        dc.w        $182,$0f00          ; color1 - Rosso
        dc.w        $184,$00f0          ; color2 - Verde
        dc.w        $186,$000f          ; color3 - Blu
        dc.w        $188,$0ff0          ; color4 - Giallo
        dc.w        $18a,$00ff          ; color5 - azzurro
        dc.w        $18c,$0fff          ; color6 - Bianco
        dc.w        $18e,$0f0f          ; color7 - viola

        dc.w        $ffff,$fffe         ; Fine della Copperlist

*****************************************************************************

        SECTION         MIOPLANE,BSS_C

Bitplanes:
        ds.b        (40*256)*3

        end

Avete visto questo lettore del corso come si da da fare? Non faccio in tempo
a finire una lezione che mi manda i suoi listati!!!
```

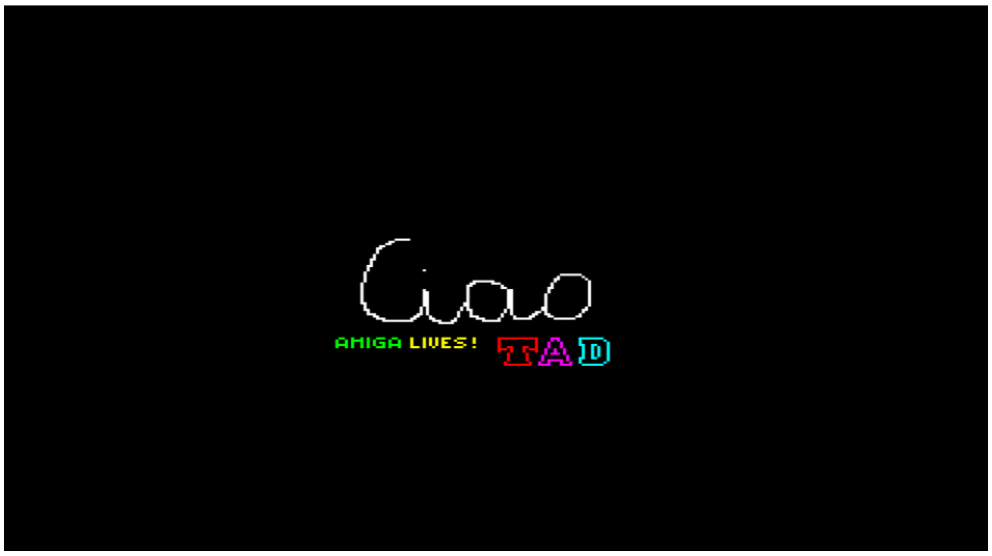


Figura 24.5: Lezione 8n4

24.13 Lezione8o

```
; Lezione8o.s        8 barre alte 13*2 linee ciascuna che rimbalzano.
;                Tasto destro per disattivare la pulizia dello sfondo.

        SECTION        Barre,CODE

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001010000000        ; solo copper DMA
;                 -----a-bcdefghij

;        a: Blitter Nasty
;        b: Bitplane DMA           (Se non e' settato, spariscono anche gli sprite)
;        c: Copper DMA
;        d: Blitter DMA
;        e: Sprite DMA
;        f: Disk DMA
;        g-j: Audio 3-0 DMA

START:
        BSR.s        INITCOPPER                ; Crea la copperlist con una routine

        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$10800,d2        ; linea da aspettare = $108
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $108
        BNE.S        Waity1

        btst        #2,$16(a5)        ; Tasto destro del mouse premuto?
        beq.s        SaltaPulizia        ; Se si non "pulire"

        BSR.s        CLRCOPPER        ; "Pulisci" lo sfondo del copper

SaltaPulizia:
        BSR.s        DOBARS                ; Fai le barre

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        rts

*****************************************************************************
*        BARRE COL COPPER - istruzioni:                                        *
*                                                                        *
*        BSR.s INITCOPPER ; eseguire prima di puntare la Copperlist per        *
```

```
*                       ; creare la copperlist (fatta di WAIT e COLOR0)*
*                                                                      *
*       BSR.s CLRCOPPER         ; eseguire per cancellare le vecchie barre       *
*                       ; "annerendo" tutti i COLOR0 della copperlist       *
*                       ; NOTA: si puo' cambiare il colore dello sfondo*
*                       ; agendo sull'equate SFONDO = $xxx               *
*                                                                      *
*       BSR.s DOBARS         ; Visualizza le barre        richiamando PUTBARS       *
*                                                                      *
************************************************************************

coplines        =       100        ; numero linee di copperlist da fare per
                              ; l'effetto delle barre.
SFONDO              =       $004        ; Colore dello "sfondo"


; INITCOPPER crea la parte di copperlist con tanti WAIT e COLOR0 di seguito

INITCOPPER:
        lea        barcopper,a0        ; Indirizzo dove creare la copperlist
        move.l        #$3001fffe,d1        ; Prima wait: linea $30 - WAIT in d1
        move.l        #$01800000,d2        ; COLOR0 in d2
        move.w  #coplines-1,d0        ; numero di linee copper
initloop:
        move.l        d1,(a0)+        ; metti il WAIT
        move.l        d2,(a0)+        ; metti il COLOR0
        add.l        #$02000000,d1        ; prossimo wait, aspetta 2 linee piu' in basso
        dbra        d0,initloop
        rts

; CLRCOPPER "pulisce" l'effetto copper, facendo diventare NERI ($000) tutti
;            i valori dei COLOR0 nella copperlist (o meglio del colore SFONDO)

CLRCOPPER:
        lea        barcopper,a0        ; Indirizzo dei WAIT/COLOR0 in copperlit
        move.w        #coplines-1,d0        ; numero di linee
        MOVE.W        #SFONDO,d1        ; Colore RGB di sfondo
clrloop:
        move.w        d1,6(a0)        ; Cambia questo Color 0
        addq.w        #8,a0                ; prossimo Color0 in copperlist
        dbra         d0,clrloop
        rts

; DOBARS effettua lo "scorrimento" delle barre colorate, una per una,
;         richiamando la sottoroutine PUTBAR per ogni barra

DOBARS:
        lea        bar1(PC),a0
        move.l        barpos1(PC),d0
        bsr.s        putbar
        move.l         d0,barpos1
        lea        bar2(PC),a0
        move.l        barpos2(PC),d0
        bsr.s        putbar
        move.l         d0,barpos2
        lea        bar3(PC),a0
        move.l        barpos3(PC),d0
        bsr.s        putbar
        move.l         d0,barpos3
        lea        bar4(PC),a0
        move.l        barpos4(PC),d0
        bsr.s        putbar
```

```
        move.l         d0,barpos4
        lea         bar5(PC),a0
        move.l         barpos5(PC),d0
        bsr.s        putbar
        move.l         d0,barpos5
        lea         bar6(PC),a0
        move.l         barpos6(PC),d0
        bsr.s        putbar
        move.l         d0,barpos6
        lea         bar7(PC),a0
        move.l         barpos7(PC),d0
        bsr.s        putbar
        move.l         d0,barpos7
        lea         bar8(PC),a0
        move.l         barpos8(PC),d0
        bsr.s        putbar
        move.l         d0,barpos8
        rts

;        Sottoroutine, in entrata:
;        a0 = indirizzo BARx, ossia i colori della barra
;        d0 = posizione BARx

putbar:
        lsl.l        #1,d0                ; sposta a sinistra di 1 bit il barpos
        lea         poslist(PC),a1        ; indirizzo tabella con le posizioni in a1
        add.l        d0,a1                ; somma barpos ad a1, trovando il giusto
                                  ; valore di posizione nella poslist
        cmp.b        #$ff,(a1)         ; siamo all'ultimo valore di poslist??
        bne.s        putbar1                ; se no, non ripartire da capo
        moveq        #0,d0
        lea         poslist,a1        ; se si, riparti da capo
putbar1:
        moveq        #0,d2
        move.b        (a1),d2                ; valore dalla tabella POSLIST
        lsl.l        #3,d2                ; shiftiamo a sinistra di 3 bit (moltipl.*8)
        lea         barcopper,a2        ; indirizzo barre in copperlist
        add.l        d2,a2                ; sommo valore preso dalla poslist e
                                  ; moltiplicato per 8, ossia trovo in a2
                                  ; l'indirizzo del wait giusto dove deve essere
                                  ; la mia barra
        moveq        #13-1,d4         ; Ogni barra e' alta 14 linee
putloop:
        move.w        (a0)+,6(a2)         ; copio il colore della barra da BARx al
                                  ; dc.w $180,xxx in copperlit
        addq.w        #8,a2                ; vado al prossimo valore di color0
        dbra         d4,putloop         ; e rifaccio 14 volte per fare tutta la barra

        lsr.l        #1,d0                ; riporto il barpos a destra di 1 bit
        addq.l        #1,d0                ; e aggiungo 1, per il prossimo ciclo.
        rts


; Queste sono le posizioni delle barre l'una rispetto all'altra. Come vedete
; sono poste l'una dopo l'altra, e in questo ordine si susseguono.

barpos1:        dc.l 0
barpos2:        dc.l 4
barpos3:        dc.l 8
barpos4:        dc.l 12
barpos5:        dc.l 16
barpos6:        dc.l 20
```

```
barpos7:        dc.l 24
barpos8:        dc.l 28


; Queste sono le 8 barre, ossia i 13 colori RGB che compongono ognuna di
; esse. Per esempio la Bar1 e' BLU, la bar2 e' GRIGIA ecc.

; colori:     RGB, RGB, RGB, RGB, RGB, RGB, RGB, RGB, RGB, RGB, RGB, RGB, RGB
bar1:
        DC.W $002,$004,$006,$008,$00a,$00c,$00f,$00c,$00a,$008,$006,$004,$002
bar2:
        DC.W $222,$444,$666,$888,$aaa,$ccc,$fff,$ccc,$aaa,$888,$666,$444,$222
bar3:
        DC.W $200,$400,$600,$800,$a00,$c00,$f00,$c00,$a00,$800,$600,$400,$200
bar4:
        DC.W $020,$040,$060,$080,$0a0,$0c0,$0f0,$0c0,$0a0,$080,$060,$040,$020
bar5:
        DC.W $012,$024,$036,$048,$05a,$06c,$07f,$06c,$05a,$048,$036,$024,$012
bar6:
        DC.W $202,$404,$606,$808,$a0a,$c0c,$f0f,$c0c,$a0a,$808,$606,$404,$202
bar7:
        DC.W $210,$420,$630,$840,$a50,$c60,$f70,$c80,$a70,$860,$650,$440,$230
bar8:
        DC.W $220,$440,$660,$880,$aa0,$cc0,$ff0,$cc0,$aa0,$880,$660,$440,$220



; Questa e' la tabella (o lista) delle posizioni verticali che possono
; assumere le barre colorate. si termina con il valore $FF.
; come indicazione, questa tabella e' fatta da "IS" con questi parametri:
; BEG>0
; END>180
; AMOUNT>150
; AMPLITUDE>85
; YOFFSET>0
; SIZE (B/W/L)>B
; MULTIPLIER>1

poslist:
        DC.B        $01,$03,$04,$06,$08,$0A,$0C,$0D,$0F,$11,$13,$14,$16,$18,$19,$1B
        DC.B        $1D,$1E,$20,$22,$23,$25,$27,$28,$2A,$2B,$2D,$2E,$30,$31,$33,$34
        DC.B        $35,$37,$38,$3A,$3B,$3C,$3D,$3F,$40,$41,$42,$43,$44,$45,$46,$47
        DC.B        $48,$49,$4A,$4B,$4C,$4D,$4D,$4E,$4F,$4F,$50,$51,$51,$52,$52,$53
        DC.B        $53,$53,$54,$54,$54,$54,$55,$55,$55,$55,$55,$55,$55,$55,$55,$55
        DC.B        $54,$54,$54,$54,$53,$53,$53,$52,$52,$51,$51,$50,$4F,$4F,$4E,$4D
        DC.B        $4D,$4C,$4B,$4A,$49,$48,$47,$46,$45,$44,$43,$42,$41,$40,$3F,$3D
        DC.B        $3C,$3B,$3A,$38,$37,$35,$34,$33,$31,$30,$2E,$2D,$2B,$2A,$28,$27
        DC.B        $25,$23,$22,$20,$1E,$1D,$1B,$19,$18,$16,$14,$13,$11,$0F,$0D,$0C
        DC.B        $0A,$08,$06,$04,$03,$01

        DC.b        $FF        ; fine della tabella

        even

*****************************************************************************
;        Copperlist
*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $8E,$2c81        ; DiwStrt
```

```
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$0038        ; DdfStart
        dc.w        $94,$00d0        ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod
        dc.w        $100,$200        ; 0 bitplanes

barcopper:                          ; Qua verra' costruita la copperlist per
        dcb.w       coplines*4,0        ; l'effetto delle barre - in questo caso
                                    ; servono 400 words. (coplines=100)

        DC.W        $ffdf,$fffe
        dc.w        $0107,$FFFE
        dc.w        $180,$222        ; Color0 grigio

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

        end
```

Questo listato mostra come si possano "costruire" delle copperlist lunghe, ma regolari, con delle routines. Piu' avanti vedremo come spesso gli effetti piu' spettacolari nascondono copperlist lunghe chilometri.

Modifiche consigliate: per rendere piu' "schiacciato" il tutto, fate attendere ogni linea, e non ogni due linee. Basta modificare INITCOPPER:

```
        add.l       #$01000000,d1        ; prossimo wait, aspetta 1 linea piu' in basso
```

Ora le barre sono alte 13 linee, e non 13*2 linee!
Potete far anche attendere ogni 3 linee, ma cosi' facendo andate troppo in basso, comunque provate:

```
        add.l       #$03000000,d1        ; prossimo wait, aspetta 3 linee piu' in basso
```

24.14 Lezionep1a

```
; Lezione8p1a.s        Funzionamento dei Condition Codes con l'istruzione MOVE

; Ecco qui il programma di questa lezione: 2 istruzioni.
; Pensate che vi stia prendendo in giro? Dopo tutto quello che avete visto
; finora pensate di sapere gia' tutto su questo semplice programma?
; Ebbene vi sbagliate. Seguite le istruzioni nel commento.

        SECTION        CondC,CODE

Inizio:
        move.w        #$0000,d0
stop:
        rts

        end

;         oO
;        \__/
;         U
```

In questa lezione e nelle successive vedremo il funzionamento dei

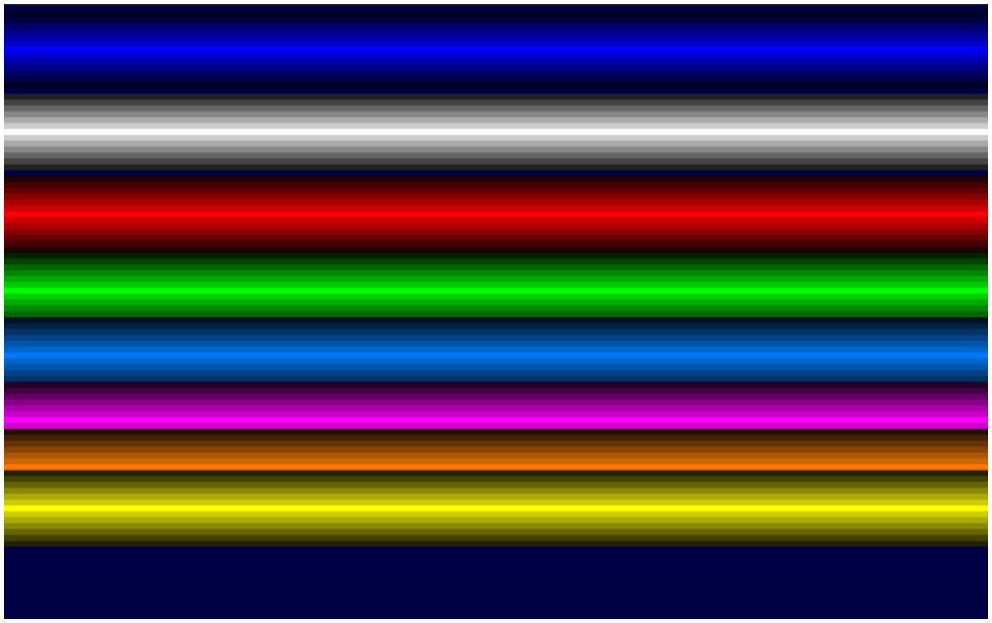


Figura 24.6: Lezione 8o

```
Condition Codes (detti CC, codici di condizione) del registro di stato.
I CC sono stati ampiamente descritti nella lezione 68000-2.TXT.
Se non vi ricordate bene cosa sono e come funzionano vi consiglio di rileggere
meglio 68000-2.TXT.
Ricordiamo brevemente che i CC sono dei bit presenti nel registro di stato che
vengono modificati dalle istruzioni assembler per dare informazioni sul
risultato dell'operazione eseguita.
Vi sono istruzioni che modificano tutti i CC, altre che ne modificano solo
alcuni e altre che non ne modificano nessuno.
Inoltre ogni istruzione che modifica i CC, lo fa in una sua propria maniera.
Nella lezione 68000-2.TXT per ogni istruzione assembler viene descritto
sinteticamente l'effetto che essa ha sui CC. In questi listati presenteremo
dei piccoli esempi pratici di come le istruzioni piu' usate modificano i CC.
Si tratta di listati piu' noiosi di quelli che avete visto sinora, ma e'
necessario che voi li studiate bene, se volte diventare dei VERI coders.
In questa lezione esamineremo l'istruzione MOVE.
Si tratta, come dovreste sapere, di un istruzione che copia il contenuto
di un registro o di una locazione di memoria e modifica di conseguenza i CC.
Per osservare bene come opera questa istruzione utilizzeremo l'ASMONE per
eseguire il programma PASSO PASSO, cioe' un'istruzione per volta.
Per farlo assemblate come al solito il programma, ma NON eseguitelo.
Date, invece all'ASMONE il comando X che serve per stampare il contenuto di
tutti i registri del 68000 e la prossima istruzione che verra' eseguita.
Queste informazioni vengono rappresentate sinteticamente dall'ASMONE nelle 4
righe riportate sotto:

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CAAE9C
SSP=07CABFD3 USP=07CAAE9C SR=0000 -- -- PL=0 ----- PC=07CAE030
PC=07CAE030 303C0000             MOVE.W  #$0000,D0
>
```

Spieghiamo brevemente il significato di queste 4 righe.
La prima riga rappresenta il contenuto degli 8 registri dati del 68000.
Potete infatti osservare come ci siano 8 numeri separati da uno spazio che
rappresentano il contenuto dei registri, cominciando da D0 (il piu' a sinistra)
e proseguendo ordinatamente fino a D7.
Notate come prima di eseguire il programma i registri siano tutti azzerati.

La seconda riga rappresenta il contenuto dei registri indirizzi, esattamente
nello stesso modo in cui la prima rappresenta il contenuto dei registri dati.
Notate che i registri sono tutti azzerati tranne A7 che contiene l'indirizzo
dello stack di sistema.

Nella terza riga sono rappresentati altri registri del processore.
Per il momento ci occuperemo soltanto del PC (Program Counter) e del SR (Status
Register)
Il PC contiene l'indirizzo della prossima istruzione da eseguire. Come sapete
le istruzioni che compongono un programma assembler si trovano in memoria!
Nel PC e' appunto contenuto l'indirizzo di memoria dal quale verra' prelevata
la prossima istruzione. In questo caso l'indirizzo e' 07CAE030, che fa parte
della memoria FAST a 32bit montata su a1200/a4000 e simili. E' ovvio che se
assemblate su diversi computer con memoria in diversi locazioni questo valore
cambiera', e anche sullo stesso computer volta per volta potra' essere diverso,
dato che i programmi sono rilocabili e non SCHIFOSAMENTE non rilocabili.

Di SR, il registro di stato abbiamo gia' parlato in 68000-2.TXT. Ci occuperemo
per ora solo del byte basso che contiene i CC. Notate che il contenuto di SR
viene rappresentato in forma esadecimale. Quindi leggere il contenuto dei
singoli CC potrebbe risultare scomodo. Per questo motivo i CC vengono
rappresentati separatamente. Noterete infatti che subito prima del contenuto
del PC ci sono 5 trattini. Ogni trattino rappresenta un diverso CC e indica
che esso e' azzerato. Quando uno dei CC assume il valore 1 al posto del
trattino viene stampata la lettera che denomina il trattino: se ad esempio
il Carry diventa 1, al posto del trattino corrispondente viene stampata la
lettera C.

Nella quarta riga infine possiamo leggere la prossima istruzione che verra'
eseguita. In questo caso si tratta della prima istruzione del programma.

NOTA: se volete stampare su un file l'output di ASMONE lo potete fare
 usando il comando > o equivalentemente selezionando la voce Output dal Menu
 command. L'ASMONE vi chiedera' il nome del file dove volete stampare l'output
 e il gioco e' fatto. E' esattamente in questo modo che sono stato stampato
 l'output del comando X

A questo punto possiamo eseguire la prima istruzione del programma, cioe'

```
          MOVE.W  #$0000,D0
```

Diamo all'ASMONE il comando K. L'istruzione verra' eseguita e ci viene stampato
automaticamente il contenuto dei registri:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CAAE9C
SSP=07CABFCF USP=07CAAE9C SR=8004 T1 -- PL=0 --Z-- PC=07CAE034
PC=07CAE034 4E75                RTS
>
```

La nostra istruzione ha messo il valore $0000 nel registro D0. Inoltre essa ha
anche variato i CC. Notate infatti che ora il contenuto di SR e' $8004, cioe'
il byte basso vale $04 che in binario si scrive %00000100. Cio' vuol dire che
il bit 2, corrispondente al CC "Zero", ha assunto il valore 1. Come vi avevo
anticipato, uno dei 5 trattini che comparivano in precedenza e' stato

```
sostituito dal Carattere "Z" che indica appunto che il Flag "Zero" ha assunto
il valore 1.
L'istruzione MOVE infatti modifica i CC nel modo seguente:
I flag V e C vengono azzerati
Il flag X non viene modificato
Il flag Z assume il valore 1 se il dato che viene copiato e' 0
Il flag N assume il valore 1 se il dato che viene copiato e' negativo.

Nel nostro caso, poiche' il dato che copiamo in D0 e' $0000, il flag Z assume
il valore 1 e il flag N assume il valore 0 (perche' $0000 NON E' un numero
negativo).

Vediamo ora qualche altro esempio di uso dell'istruzione MOVE. Modificate nel
sorgente la MOVE, scrivendo:

        move.w        #$1000,d0

Ripetete ora la procedura per eseguire il programma PASSO PASSO.
Dopo aver eseguito la MOVE avremo la seguente situazione:

D0: 00001000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07C9EDFC
SSP=07C9FF2F USP=07C9EDFC SR=8000 T1 -- PL=0 ----- PC=07CA2E40
PC=07CA2E40 4E75                RTS

Possiamo notare che ora D0 contiene il valore $00001000, ovvero proprio
quello che gli abbiamo copiato noi con la MOVE. Inoltre questa volta i CC
sono tutti azzerati. Cio' dipende dal fatto che il valore $1000 che noi
abbiamo spostato e' diverso da zero ed inoltre e' un numero positivo.

Facciamo ora un'altra modifica.
Invece del valore $1000 mettiamo $8020, ottenendo:

        move.w        #$8020,d0        ; ossia "move.w #-32736,d0

Questa volta dopo l'esecuzione della MOVE otteniamo:

D0: 00008020 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07C9EDFC
SSP=07C9FF2F USP=07C9EDFC SR=8008 T1 -- PL=0 -N--- PC=07CA2E40
PC=07CA2E40 4E75                RTS

Come potete vedere, D0 ha assunto il valore desiderato e il flag N ha assunto
valore 1. Cio' dipende dal fatto che il numero $8020 e' un numero negativo
perche' il suo bit piu' significativo vale 1.

Trasformiamo ora la MOVE come segue:

        move.l        #$8020,d0

Abbiamo semplicemente cambiato la dimensione del dato spostato. Questo fatto
comporta che ora dobbiamo considerare il valore $8020 come un numero a 32 bit
ovvero come $00008020. Ora il bit piu' significativo e' il bit 31, non il
bit 15 come in precedenza! Quindi in questo caso abbiamo a che fare con un
numero POSITIVO. Eseguendo la MOVE si ottiene infatti:

D0: 00008020 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07C9EDC4
SSP=07C9FEF7 USP=07C9EDC4 SR=8000 T1 -- PL=0 ----- PC=07CA33CA
PC=07CA33CA 4E75                RTS
>
```

dove potete notare che il flag "N" e‘ azzerato.

Lezionep1b

```
; Lezione8p1b.s                Funzionamento dei Condition Codes con le MULU/MULS

        SECTION        CondC,CODE

;         oO
;         C _
;        \__/
;          U

Inizio:
        move.l        #$0003,d0        ; Sarebbe piu' veloce un "moveq #3,d0"...
        move.l        #$c000,d1
        muls.w        d0,d1

        moveq        #3,d0                ; Qua lo abbiamo usato... vah!
        move.l        #$c000,d1
        mulu.w        d0,d1
stop:
        rts

        end
```

Vediamo ora un esempio d'uso delle istruzioni di moltiplicazione.
Il 68000 ci mette a disposizione 2 diverse istruzioni di moltiplicazione:
MULS moltiplica 2 numeri considerandoli come numeri in complemento a 2,
mentre MULU considera i numeri da moltiplicare sempre positivi.
Muls/Divs lavorano con numeri in complemento a due mentre Mulu/Divu usano
numeri senza segno.

```
        MULU    <ea>,Dn         Sorgente=Dati    Destinazione=Dn
        MULS    <ea>,Dn         Sorgente=Dati    Destinazione=Dn
```

E' possibile moltiplicare solo numeri a 16 bit (in formato di word) e
il prodotto a 32bit (formato di longword) e' fornito in un registro dati.
Ovviamente i risultati che si ottengono con MULU o MULS sono molto diversi.
Facciamo un esempio moltiplicando $c000 per $0003.

```
D0: 00000003 0000C000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8000 T1 -- PL=0 ----- PC=07D34CEC
PC=07D34CEC C3C0                MULS.W  D0,D1
>
```

La MULS considera $c000 come un numero negativo.
Il risultato che ottiene e‘ il seguente:

```
D0: 00000003 FFFF4000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8008 T1 -- PL=0 -N--- PC=07D34CEE
PC=07D34CEE 203C00000003        MOVE.L  #$00000003,D0
>
```

Il risultato e‘ negativo (infatti abbiamo moltiplicato un numero positivo per
un negativo) e pertanto il flag N vale 1.
Ricordo agli ignoranti che se si moltiplicano due numeri positivi tra loro il
risultato e' positivo, allo stesso modo se si moltiplicano 2 numeri negativi

```
tra loro il risultato e' positivo. Invece se si moltiplica un numero negativo
per uno positivo, o uno positovo per uno negativo, il risultato e' negativo.
In sintesi:         + * + = +       - * - = +       + * - = -       - * + = -
Vediamo ora come si comporta la MULU, che considera $c000 come un numero
positivo.

D0: 00000003 0000C000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8000 T1 -- PL=0 ----- PC=07D34CFA
PC=07D34CFA C2C0                MULU.W  D0,D1
>

Il risultato che si ottiene e‘ il seguente:

D0: 00000003 00024000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8000 T1 -- PL=0 ----- PC=07D34CFC
PC=07D34CFC 4E75                RTS
>

Come potete vedere esso e‘ molto diverso. Tra l'altro e‘ positivo, e infatti
il flag N vale 0. Quindi anche per quanto riguarda le moltiplicazioni si deve
scegliere con attenzione l'istruzione da impiegare.
```

Lezionep1c

```
; Lezione8p1c.s                Funzionamento dei Condition Codes con le DIVU/DIVS

        SECTION        CondC,CODE

Inizio:
        moveq         #$0010,d0
        moveq         #$0003,d1
        divs.w         d1,d0

        move.l         #$200000,d0
        moveq         #$0002,d1
        divs.w         d1,d0
stop:
        rts

        end

;        ·[oO]·
;          C
;         \__/
;           U

Vediamo ora un esempio d'uso delle istruzioni di divisione.
Anche per la divisione ll 68000 ci mette a disposizione 2 diverse istruzioni:
DIVS divide 2 numeri considerandoli come numeri in complemento a 2,
mentre DIVU considera i numeri da dividere sempre positivi.
Le differenze quindi sono analoghe a quelle tra MULS e MULU, pertanto non le
illustreremo, fate degli esperimenti per esercizio.
Gli esempi che faremo riguardano la DIVS.
Le istruzioni di divisione dividono un operando a 32bit in un registro dati
con un divisore a 16 bit, il quoziente a 16 bit sara' messo nella word bassa
del registro destinazione e il resto nella word alta.
Nel caso di divisione per 0, il 68000 effettuera' una routine di eccezione,
e nella maggior parte dei casi si avra' una bella GURU MEDITATION.
```

```
La divisione puo' influenzare i codici condizione in questo modo:

1) Carry (C) e' sempre posto a 0
2) Overflow (V) e' settato se il dividendo e' tanto maggiore del divisore che
il risultato non puo' essere contenuto in 16bit
es:
        move.l        #$ffffffff,d0
        divu.w        #2,d0

3) Zero (Z) e' posto a 1 se il risultato dell'operazione e' 0
4) Negativo (N) e' posto a 1 se il risultato dell'operazione e' negativo
5) Extend (X) rimane invariato.
-------------------------------------------------------------------------------

Prima vediamo un esempio normale: dividiamo il numero $10(=16) contenuto nel
registro D0 per il numero 3, contenuto in D1.

D0: 00000010 00000003 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8000 T1 -- PL=0 ----- PC=07D34CE8
PC=07D34CE8 81C1                DIVS.W  D1,D0
>

Il risultato e‘ riportato sotto. Notate che vengono calcolati sia il quoziente
(posto nella word bassa D0) che il resto (posto nella word alta di D0).
Si tratta infatti di una divisione tra interi.

D0: 00010005 00000003 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8000 T1 -- PL=0 ----- PC=07D34CEA
PC=07D34CEA 203C00200000        MOVE.L  #$00200000,D0
>

Vediamo ora un'altro esempio.
Dividiamo il numero $200000 (contenuto in D0) per $2 (in D1).

D0: 00200000 00000002 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8000 T1 -- PL=0 ----- PC=07D34CF6
PC=07D34CF6 81C1                DIVS.W  D1,D0
>

Il risultato esatto e‘ $100000, come potete verificare con il comando "?" di
ASMONE. Questo numero pero‘ e‘ troppo grande per essere contenuto in una word.
Pertanto la DIVS non riesce a effettuare il calcolo correttamente e segnala
questo fatto ponendo a 1 il flag V:

D0: 00200000 00000002 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07D32154
SSP=07D33287 USP=07D32154 SR=8002 T1 -- PL=0 ---V- PC=07D34CF8
PC=07D34CF8 4E75                RTS
>

In casi come questo si deve effetture la divisione mediante algoritmi
particolari.
```

24.15 Lezionep2a

```
; Lezione8p2a.s                Flag e registri indirizzi
```

```
        SECTION        CondC,CODE

Inizio:
        move.w         #$8000,d0
        move.l         #$80000000,a0
stop:
        rts

        end


;           . . . .
;          .       .
;          .       .
;           .     .
;             . .
```

In questa lezione ci occuperemo di una particolarita‘ dell’indirizzamento diretto a registro indirizzi. Vedremo questa particolarita‘ utilizzando una istruzione MOVE nella quale per la destinazione e‘ usato l’indirizzamento diretto a registro indirizzi, ma essa si verifica con tutte le istruzioni che ammettono l’indirizzamento diretto a registro indirizzi per la destinazione.

Per prima cosa assemblate il programma ed eseguite la prima istruzione. Otterrete il seguente output:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 80000000 00000000 00000000 00000000 00000000 00000000 00000000 07C9EDC4
SSP=07C9FEF7 USP=07C9EDC4 SR=8004 T1 -- PL=0 --Z-- PC=07CA18DC
PC=07CA18DC 207C80000000          MOVE.L  #$80000000,A0
>
```

Come ci aspettavamo il flag "Z" ha assunto il valore 1.
Eseguiamo anche la seconda istruzione:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 80000000 00000000 00000000 00000000 00000000 00000000 00000000 07C9EDC4
SSP=07C9FEF7 USP=07C9EDC4 SR=8004 T1 -- PL=0 --Z-- PC=07CA18E2
PC=07CA18E2 4E75                  RTS
>
```

Notiamo che l’istruzione e‘ stata eseguita, ma che il flag "Z" vale ancora 1 e il flag "N" vale invece 0. Eppure il valore $80000000 che abbiamo caricato nel registro A0 e‘ negativo! Dunque il nostro fido 680x0 si e‘ sbagliato? Ma naturalmente no! (Mica e‘ un Pentium 60! :). Il punto e‘ che come abbiamo gia‘ spiegato nella lezione 8, in realta‘ l’istruzione che si occupa di copiare dati in un registro indirizzi e‘ la MOVEA, una variante della normale MOVE; l’ASMONE per comodita‘ ci consente di scrivere MOVE per copiare nei registri indirizzi, e si occupa lui di sostituire la MOVE con la MOVEA. Di solito noi non ci accorgiamo nemmeno di questa sostituzione. In questo caso pero‘ bisogna stare molto attenti perche‘ la MOVEA si comporta diversamente dalla MOVE per quanto riguarda la modifica dei CC. La MOVEA, come potete leggere in 68000-2.TXT, lascia i CC TUTTI INALTERATI. Nel nostro caso, il flag "Z" valeva 1 prima dell’esecuzione della MOVE #$80000000,A0 e per questo motivo e‘ rimasto al valore 1. Verifichiamolo modificando la prima MOVE in

```
        move.w         #$8000,d0
```

Eseguendo PASSO PASSO notiamo che la prima MOVE fa assumere il valore 1 al flag "N":

```
D0: 00008000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CC685C
SSP=07CC798F USP=07CC685C SR=8008 T1 -- PL=0 -N--- PC=07CC9A60
PC=07CC9A60 207C80000000         MOVE.L  #$80000000,A0
>
```

E la MOVE.L #$80000000,A0 come abbiamo detto lascia inalterati i CC:

```
D0: 00008000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 80000000 00000000 00000000 00000000 00000000 00000000 00000000 07CC685C
SSP=07CC798F USP=07CC685C SR=8008 T1 -- PL=0 -N--- PC=07CC9A66
PC=07CC9A66 4E75                 RTS
>
```

Bisogna prestare particolare attenzione al fatto che i CC non vengono influenzati quando si indirizzano i registri indirizzi, perche‘ questo fatto puo‘ essere causa di BUG. Supponiamo per esempio di avere memorizzato un dato e di volerlo modificare in due modi diversi a seconda se esso sia positivo o negativo. Se spostiamo il dato in un registro dati, per esempio D0, possiamo scrivere il seguente frammento di codice:

```
        move.w        dato(pc),d0        ; modifica i CC in base al dato
        bmi.s        dato_negativo
dato_positivo:
 ; operazioni da compiere se il dato e‘ positivo
        bra.s        fine

dato_negativo:
 ; operazioni da compiere se il dato e‘ negativo
fine:
        ; resto del programma
```

In questo caso come gia‘ sappiamo la MOVE provvede a settare i CC a seconda del segno del dato.
Se invece dovessimo mettere il nostro dato in un registro indirizzi (es. A0) se scrivessimo una procedura analoga non funzionerebbe perche‘ la MOVEA non aggiorna correttamente i CC.

```
        move.w        dato(pc),a0        ; NON modifica i CC in base al dato !!
        bmi.s        dato_negativo        ; Il salto viene effettuato in base allo
                                ; stato dei CC precedente alla MOVE
dato_positivo:
        ; operazioni da compiere se il dato e‘ positivo
        bra.s        fine

dato_negativo:
        ; operazioni da compiere se il dato e‘ negativo

fine:
        ; resto del programma
```

Una possibile soluzione al problema potrebbe essere di spostare il dato prima in un registro dati e successivamente in A0, oppure usare l’istruzione TST.

Lezionep2b

```
;        Lezione8p2b.s        estensione del segno nei registri indirizzi

        SECTION        CondC,CODE
```

```
Inizio:
        move.l        #$ffffffff,a0        ; Ossia "move.l #-1,a0"
        move.w        #$51a7,a0
stop:
        rts

        end

;            \|/
;           (©_©)
;--------ooO-(_)-Ooo--------
```

In questa lezione ci occuperemo di un'altra particolarita' dell'indirizzamento diretto a registro indirizzi.
Eseguiamo un'istruzione per volta il programma sopra riportato.
La prima MOVE carica un valore di 32 bit in A0.

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: FFFFFFFF 00000000 00000000 00000000 00000000 00000000 07C9F584
SSP=07CA06B7 USP=07C9F584 SR=8000 T1 -- PL=0 ----- PC=07CA1F8E
PC=07CA1F8E 307C0100                MOVE.W  #$51A7,A0
>
```

Come normale il registro A0 ha assunto il valore $FFFFFFFF. Ora eseguiamo la seconda MOVE. Notiamo che essa carica un valore a 16 bit in A0.
Ci aspetteremmo che solo la word bassa di A0 venisse modificata.
Invece possiamo verificare che e' stata modificata anche la word alta:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 000051A7 00000000 00000000 00000000 00000000 00000000 07C9F584
SSP=07CA06B7 USP=07C9F584 SR=8000 T1 -- PL=0 ----- PC=07CA1F92
PC=07CA1F92 4E75                RTS
>
```

Questo accade perche' quando si scrive in un registro indirizzi una WORD (ricordiamo che NON e' possibile scrivere un singolo BYTE, cioe' l'istruzione MOVE.B xxxx,Ax NON e' permessa) essa viene trasformata in una LONG WORD mediante un'operazione detta "estensione di segno" che consiste nel copiare il bit piu' significativo della WORD (cioe' il bit 15, che come sapete indica il segno di un valore formato WORD) in tutti i bit della WORD alta, in modo da conservare lo stesso segno passando dal valore WORD a quello LONG WORD. In pratica nel nostro caso abbiamo:

```
valore di partenza = $51A7 = %0101000110100111
                              ^
                              |
                             bit piu' significativo vale 0

valore esteso = $000051A7  = %00000000000000000101000110100111
```

tutti i bit da 16 a 31 hanno assunto il valore 0.

Facciamo un'altro esempio, cambiando i valori caricati dalle MOVE:

```
        move.l        #$22222222,a0
        move.w        #$c1a7,a0
```

Eseguendo la prima MOVE otteniamo:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 22222222 00000000 00000000 00000000 00000000 00000000 07C9F584
SSP=07CA06B7 USP=07C9F584 SR=8000 T1 -- PL=0 ----- PC=07CA2642
```

```
PC=07CA2642 307CC1A7                MOVE.W  #$C1A7,A0
>
```

eseguendo la seconda:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: FFFFC1A7 00000000 00000000 00000000 00000000 00000000 00000000 07C9F584
SSP=07CA06B7 USP=07C9F584 SR=8000 T1 -- PL=0 ----- PC=07CA2646
PC=07CA2646 4E75                RTS
```

In questo caso l'estensione di segno ha reso negativo il valore LONG WORD:

```
Valore di partenza = $C1A7 = %1100000110100111
                              ^
                              |
                              Il bit piu‘ significativo vale 1
Valore esteso = $FFFFC1A7  = %11111111111111111100000110100111
```

Tutti i bit da 16 a 31 hanno assunto il valore 1.

Nota: l'istruzione EXT.L serve ad estendere il segno come in questi esempi.

24.16 Lezionep3

```
; Lezione8p3.s        Funzionamento dei Condition Codes con l'istruzione TST

        SECTION        CondC,CODE

Inizio:
        tst.w        dato
stop:
        rts


dato:
        dc.w        $ff02

        end

;         \  /
;          oO
;         \__/
```

L'istruzione tst in pratica confronta l'operando con zero.
Abbiamo visto che l'istruzione MOVE modifica i CC dandoci informazioni sul
dato che viene copiato. Se vogliamo ottenere quelle informazioni SENZA copiare
il dato, possiamo usare l'istruzione TST.
Si ratta di un'istruzione ad un solo operando, che legge un valore e modifica
tutti i CC in base ad esso.
I CC vengono modificati allo stesso modo dell'istruzione MOVE:

I flag V e C vengono azzerati
Il flag X non viene modificato
Il flag Z assume il valore 1 se il dato testato e‘ 0
Il flag N assume il valore 1 se il dato testato e‘ negativo.

Assemblate il programma e eseguite l'istruzione TST:

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CBD594
```

```
SSP=07CBE6C7 USP=07CBD594 SR=8008 T1 -- PL=0 -N--- PC=07CC0F52
PC=07CC0F52 4E75                RTS
>

Il flag N ha assunto il valore 1 perche‘ la WORD all’indirizzo di
memoria "dato" vale $ff03 che e‘ un numero negativo perche‘ il suo bit piu‘
significativo vale 1.

Potete variare il valore contenuto all’indirizzo "dato" e osservare come
si comporta il TST.
Notate che non e‘ possibile usare il TST con registri indirizzi, cioe‘
se tentate di assemblare

        TST.W        A0

L’ASMONE vi dara‘ un messaggio di errore.
```

24.17 Lezionep4

```
; Lezione8p4.s        Funzionamento dei Condition Codes con le istruzioni
;                logiche AND, NOT, OR, EOR

        SECTION        CiriCop,CODE

Inizio:
        not.w        dato1
        not.w        dato2
        move.w        #$ff00,d0
        and.w        dato1,d0
        move.w        #$0003,d0
        and.w        dato2,d0
        move.w        #$8000,d0
        or.w        dato1,d0
        move.w        #$8000,d0
        eor.w        d0,dato3
stop:
        rts

dato1:
        dc.w        $ff00
dato2:
        dc.w        $0f00
dato3:
        dc.w        $c000

        end

;        .----------.
;        |   \||/   |
;        |   (oo)   |
;        ‘-oO-\/-Oo-’

Le istruzioni logiche modificano i CC allo modo analogo alle istruzioni MOVE
e TST, ovvero:

I flag V e C vengono azzerati
Il flag X non viene modificato
Il flag Z assume il valore 1 se il risultato dell’operazione e‘ 0
Il flag N assume il valore 1 se il risultato dell’operazione e‘ negativo.
```

```
Lo verifichiamo eseguendo PASSO PASSO il nostro programma, nel quale
presentiamo diversi esempi.

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA304B USP=07CA1F14 SR=0000 -- -- PL=0 ----- PC=07CA4AF4
PC=07CA4AF4 467907CA4B2A         NOT.W   $07CA4B2A
>

La prima istruzione da eseguire e‘ una NOT. Il numero $7CA4B2A e‘ l’indirizzo
"dato1" (naturalmente a voi risultera’ un’altra locazione!).

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8000 T1 -- PL=0 ----- PC=07CA4AFA
PC=07CA4AFA 467907CA4B2C        NOT.W   $07CA4B2C
>

Il risultato dell’operazione lo possiamo vedere con il comando "M.W DATO1" ,
ed e‘ $00ff. (nota: il comando "m" puo’ essere anche "m.w" o "m.l" per mostrare
una word o una longword alla volta).
Si ratta di un numero positivo diverso da zero, pertanto i flag Z e N sono
azzerati. Eseguiamo ora la seconda NOT:

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8008 T1 -- PL=0 -N--- PC=07CA4B00
PC=07CA4B00 303CFF00                MOVE.W  #$FF00,D0
>

Questa volta il risultato e‘ negativo (andate a guardare all’indirizzo DATO2)
e infatti il flag N risulta settato. Ora carichiamo un valore in D0.

D0: 0000FF00 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8000 T1 -- PL=0 ----- PC=07CA4B04
PC=07CA4B04 C07907CA4B2A         AND.W   $07CA4B2A,D0
>

E facciamo l’AND con il valore "DATO1"

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8004 T1 -- PL=0 --Z-- PC=07CA4B0A
PC=07CA4B0A 303C0003                MOVE.W  #$0003,D0
>

Il risultato e‘ zero, e pertanto il flag Z assume il valore 1.
Ora carichiamo un nuovo valore in D0 e facciamo l’ AND con DATO2.

D0: 00000003 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8000 T1 -- PL=0 ----- PC=07CA4B0E
PC=07CA4B0E C07907CA4B2C         AND.W   $07CA4B2C,D0
>

Questa volta otteniamo un risultato positivo, diverso da zero.
Ora passiamo all’OR. Prima carichiamo un valore negativo in D0.

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8000 T1 -- PL=0 ----- PC=07CA4B14
```

```
PC=07CA4B14 303C8000             MOVE.W  #$8000,D0
>
```

E poi effettuiamo l'OR con "DATO1"

```
D0: 00008000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8008 T1 -- PL=0 -N--- PC=07CA4B18
PC=07CA4B18 807907CA4B2A         OR.W    $07CA4B2A,D0
>
```

Come vedete otteniamo un valore che e' ancora negativo, perche' il bit
piu' significativo ha ancora il valore 1.

```
D0: 000080FF 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8008 T1 -- PL=0 -N--- PC=07CA4B1E
PC=07CA4B1E 303C8000                MOVE.W  #$8000,D0
>
```

Ora un'ultima prova. Carichiamo ancora $8000 in D0:

```
D0: 00008000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8008 T1 -- PL=0 -N--- PC=07CA4B22
PC=07CA4B22 B17907CA4B2E         EOR.W   D0,$07CA4B2E
```

E facciamo l'EOR con "DATO3":

```
D0: 00008000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA1F14
SSP=07CA3047 USP=07CA1F14 SR=8000 T1 -- PL=0 ----- PC=07CA4B28
PC=07CA4B28 4E75                 RTS
>
```

Questa volta otteniamo un risultato positivo e diverso da zero, come potete
verificare voi stessi.

24.18 Lezionep5

```
; Lezione8p5.s        Funzionamento dei Condition Codes con l'istruzione NEG

        SECTION        CondC,CODE

Inizio:
        neg.w        dato1
        neg.w        dato2
        neg.w        dato3
        neg.w        dato4
stop:
        rts

dato1:
        dc.w        $ff02
dato2:
        dc.w        $4f02
dato3:
        dc.w        $0000
dato4:
        dc.w        $8000
```

```
        end

Vediamo ora un esempio sull'istruzione NEG.
Vi sono due istruzioni di negazione che consentono di complementare a 2 un
operando .B .W o .L sotrraendolo da 0.
-------------------------------------------------------------------------
NEG     <ea>            Sorgente=All
NEGX    <ea>            Sorgente=All
-------------------------------------------------------------------------
L'istruzione di negazione puo' influenzare cosi' i codici condizioni:

1.Bit0, Carry (C): e' posto a 0 se l'operando e' zero, altrimenti e' posto a 1.

2.Bit1, Overflow (V): Il bit e' posto a 1 solo se l'operando ha il valore di
$80 byte, $8000 word, $80000000 long.

3.Bit2, Zero (Z): Il bit e' posto a 1 se il risultato dell'operazione e' zero.
4.Bit3, Negativo (N):  e' posto a 1 se l'operando e' un numero positivo diverso
da zero.
5.Bit4, Extend (X): assume lo stesso stato del bit C
-------------------------------------------------------------------------------

La prima istruzione del listato opera sul dato all'indirizzo "DATO1", che e'
un numero negativo. Eseguendola otteniamo:

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA7934
SSP=07CA8A67 USP=07CA7934 SR=8011 T1 -- PL=0 X---C PC=07CFEBDA
PC=07CFEBDA 447907CFEBF0         NEG.W   $07CFEBF0
>

Come potete constatare con il comando ASMONE "M.w dato1" il risultato e'
positivo e' diverso da zero. Pertanto gli unici CC ad essere settati a 1
sono C ed X.
La seconda NEG opera invece su un dato positivo. Il risultato e' quindi
negativo, e di conseguenza questa volta anche il bit N vale 1:

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA7934
SSP=07CA8A67 USP=07CA7934 SR=8019 T1 -- PL=0 XN--C PC=07CFEBE0
PC=07CFEBE0 447907CFEBF2         NEG.W   $07CFEBF2
>

Siamo ora alla terza NEG, che opera sul valore contenuto all'indirizzo
"dato3" che e' zero. Come potete verificare il risultato e' ancora zero,
perche' giustamente il negativo (e quindi il complemento a 2) di zero e'
ancora zero. Per quanto riguarda i CC, sono tutti azzerati tranne Z:

D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA7934
SSP=07CA8A67 USP=07CA7934 SR=8000 T1 -- PL=0 --Z-- PC=07CFEBE6
PC=07CFEBE6 447907CFEBF4         NEG.W   $07CFEBF4
>

Veniamo ora all'ultimo caso. Il valore su cui opera la NEG stavolta e'
$8000 = -32678. Come sapete con 16 bit NON possiamo rappresentare il valore
+32678. Poiche' in questo caso la NEG opera su word, essa non puo' calcolare
correttamente il risultato che cerchiamo. Eseguendola, vediamo che essa
lascia INALTERATO (cioe' a $8000) il valore contenuto all'indirizzo "dato4"
e assegna al flag V (oVerflov) il valore 1 per segnalarci l'errore:
```

```
D0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA7934
SSP=07CA8A67 USP=07CA7934 SR=801B T1 -- PL=0 XN-VC PC=07CFEBEC
PC=07CFEBEC 4E75                RTS
>
```

24.19 Lezionep6

```
; Lezione8p6.s        Funzionamento dei Condition Codes con l'istruzione ADD

        SECTION        CondC,CODE

Inizio:
        move.w         #$4000,d0
        move.w         #$2000,d1
        add.w         d0,d1
        move.w         #$e000,d0
        move.w         #$b000,d1
        add.w         d0,d1
        move.w         #$6000,d0
        move.w         #$5000,d1
        add.w         d0,d1
        move.w         #$9000,d0
        move.w         #$a000,d1
        add.w         d0,d1
stop:
        rts

        end


L'istruzione ADD influenza i codici condizione cosi':

1) Bit0, Carry (C): e' posto a 1 se il risultato non puo' essere contenuto
   nell'operando destinazione.
   Esempio: (si assume che i numeri siano senza segno.)

        move.w         #$7001,d0         ; d0=$7001
        add.w         #$8fff,d0         ; d0=$7001+$8fff=$10000

  Come si puo' vedere il risultato dell'addizione non puo' essere contenuto in
  una word in quanto servirebbero 17 bit, il flag C viene settato.

2) Bit1, Overflow (V): Il bit e' posto a 1 solo se l'addizione di due numeri
   con lo stesso segno supera come risultato il campo in complemento a 2
   dell'operando (per es. nel caso di operandi WORD V vale 1 se il risultato
   e' maggiore di 32767 oppure se e' minore di -32768)
   Esempio: (numeri con segno)

        move.w         #$7fff,d0         ; d0=$7fff
        addq.w         #$1,d0                 ; d0=$7fff+1=$8000=-32768 !!!!!

  in questo caso il bit di Overflow viene settato.

3) Bit2, Zero (Z): Il bit e' posto a 1 se il risultato dell'operazione e' zero.
4) Bit3, Negativo (N): Il bit e' posto a 1 se l'ultima operazione ha prodotto
   un risultato negativo.
5) Bit4, Extend (X): assume lo stesso stato del bit C

V ed N hanno senso solo se si sommano numeri con segno.
```

```
N.B.: Se le operazioni hanno come operando destinazione un registro indirizzi,
i codici condizione rimangono INVARIATI!!!!
Questa e' una variazione dell'istruzione ADD, ed e' chiamata add address ADDA.
--------------------------------------------------------------------------------

Ora verifichiamo la teoria.
Eseguiamo i primi 2 passi del programma: si tratta di 2 MOVE che hanno
come effetto quello di caricare i 2 valori che vogliamo sommare in 2
registri. Si tratta di 2 valori positivi.

D0: 00004000 00002000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4754
SSP=07CA5887 USP=07CA4754 SR=8000 T1 -- PL=0 ----- PC=07CA7A74
PC=07CA7A74 D240                ADD.W   D0,D1
>

Eseguiamo la somma. Come potete verificare "a mano", questa somma non genera
riporti, ovvero il risultato ($6000) e' un numero minore di $7fff, e pertanto
puo' essere contenuto ancora in una word. Quindi i flag C,X e V vengono
azzerati. Inoltre anche Z ed N vengono azzerati, perche' $6000 e' positivo e
diverso da zero.

D0: 00004000 00006000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4754
SSP=07CA5887 USP=07CA4754 SR=8000 T1 -- PL=0 ----- PC=07CA7A76
PC=07CA7A76 303CE000                MOVE.W  #$E000,D0
>

Facciamo ora una somma tra $e000 e $b000. In questo caso abbiamo a che fare
con numeri negativi. Il risultato (che potete verificare a mano)
e' $9000=-28672 che e' maggiore di -32768 e quindi non da problemi, per cui
il flag V vale zero.
Notate pero' che volendo potremmo considerare i nostri 2 numeri come positivi
tralasciando il segno. In questo caso, cioe', le nostre word assumerebbero
valori compresi tra 0 e 65535. In questo caso, il risultato che otteniamo,
cioe' $9000 non e' ovviamente corretto. Cio' accade perche' il risultato esatto
di $e000+$b000 (considerarti come positivi) sarebbe $19000=102400, cioe' un
numero maggiore di 65535 che avrebbe bisogno di 17 bit per essere
rappresentato correttamente.
Il 68000, per ovviare a questo problema, memorizza il 17-esimo bit nel Carry,
(e anche in X) che quindi assume valore 1. Notate inoltre che siccome $9000
e' negativo (considerato in complemento a 2) anche il flag N assume valore 1.
Ecco quindi cio' che ottenete eseguendo la somma:

D0: 0000E000 00009000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4754
SSP=07CA5887 USP=07CA4754 SR=8019 T1 -- PL=0 XN--C PC=07CA7A80
PC=07CA7A80 303C6000                MOVE.W  #$6000,D0
>

Vediamo un terzo esempio. Questa volta sommiamo $5000(=20480) e $6000(=24576).
Si tratta di 2 numeri positivi. A differenza del primo esempio, pero',
se eseguiamo la somma a mano vediamo che il risultato e' 45056(=$b000) che
risulta maggiore di 32767, e infatti come potete notare e' un numero negativo.
Pertanto se interpretiamo i numeri in complemento a 2 (cioe' vanno da -32768
a 32767) il risultato e' sbagliato, e pertanto il flag V assume valore 1.
Se invece interpretiamo i numeri come sempre positivi (cioe' da 0 a 65536)
il risultato e' corretto, perche' e' minore di 65535. Pertanto il flag C
assume valore zero. Il flag N assume comunque il valore 1 perche' abbiamo un
numero negativo (se interpretato in complemento a 2). Infatti:
```

```
D0: 00006000 0000B000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4754
SSP=07CA5887 USP=07CA4754 SR=800A T1 -- PL=0 -N-V- PC=07CA7A8A
PC=07CA7A8A 303C9000            MOVE.W  #$9000,D0
>
```

Vediamo ora un'ultimo esempio. Sommiamo $9000 e $a000. Si tratta di 2 numeri
negativi. Se li interpretiamo in complemento a 2 e li sommiamo, notiamo
che il risultato e' minore di -32768. Pertanto il flag V assume valore 1.
Se li interpretiamo come numeri positivi, abbiamo che la loro somma sarebbe
$13000 che ha bisogno di 17 bit. Pertanto anche il flag C vale 1.
Come risultato otteniamo $3000 ovvero i 16 bit meno significativi della somma.
Poiche' $3000 e' positivo, il flag N vale zero.

```
D0: 00009000 00003000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4754
SSP=07CA5887 USP=07CA4754 SR=8013 T1 -- PL=0 X--VC PC=07CA7A94
PC=07CA7A94 4E75                RTS
>
```

24.20 Lezionep7

```
; Lezione8p7.s        Funzionamento dei Condition Codes con l'istruzione CMP

        SECTION        CondC,CODE

Inizio:
        move.w        #$9000,d0
        move.w        #$6000,d1
        cmp.w        d0,d1
        bgt.w        salto
stop:
        rts

salto:
        nop        ; questo salto viene effettuato se la destinazione e' maggiore
                ; della sorgente
        rts

        end
```

L'istruzione CMP ci permette di confrontare 2 numeri e settare di conseguenza
i CC. Di solito una CMP e' seguita da un'istruzione Bcc. Ecco i 3 "tipi":

```
CMPA.x        <ea>,Ay                Sorgente=All        Destinazione=An (nota: SOLO .W o .L).
-------------------------------------------------------------------------------
CMPI.x        #d,<ea>                Sorgente=#d        Destinazione=Dati alterabili
-------------------------------------------------------------------------------
CMPM.x        (Ax)+,(Ay)+        Sorgente=(An)+        Destinazione=(An)+
-------------------------------------------------------------------------------
```

Ognuna delle istruzioni di confronto del 68000 sottrae l'operando Sorgente
dalla Destinazione e setta i flag di condizione secondo la seguente tabella:

```
+----------------------+---+---+---+---+
|Condizione            | N | Z | V | C |
+----------------------+---+---+---+---+
|Sorgente<Destinazione | 0 | 0 |0/1| 0 |
+----------------------+---+---+---+---+
|Sorgente=Destinazione | 0 | 1 | 0 | 0 |
```

```
+----------------------+---+---+---+---+
|Sorgente>Destinazione | 1 | 0 |0/1| 1 |
+----------------------+---+---+---+---+
```

Il bit V vale 1 se la differenza tra sorgente e destinazione supera come
risultato il campo in complemento a 2 dell'operando (cioe' se e' minore del
piu' piccolo numero negativo rappresentabile o maggiore del piu' grande
positivo rappresentabile).
N e V sono significativi solo se si confrontano operandi in complemeto a 2.

N.B.: Diversamente dalle istruzione di sottrazione, le istruzioni di confronto
non salvano il risultato della sottrazione!!!!!!!! (Mi pare chiaro!)

Le Bcc leggono lo stato dei CC e nel caso sia verificata una particolare
condizione (che varia tra le singole Bcc) eseguono o no un salto.
La CMP setta i flag CC allo stesso modo della SUB.
Vediamo un breve esempio. Effettuiamo un confronto tra un numero positivo
e uno negativo. Vediamo che l'esito del confronto e' diverso se consideriamo
il numero negativo come positivo.
Eseguiamo il programma fino all'istruzione BGT.

```
D0: 00009000 00006000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4F64
SSP=07CA6097 USP=07CA4F64 SR=800B T1 -- PL=0 -N-VC PC=07CA7B52
PC=07CA7B52 6E000004                BGT.W   $07CA7B58
>
```

La BGT come sapete effettua il salto se l'operando destinazione e' maggiore
dell'operando sorgente.
Inotre essa considera i numeri come valori in complemento a 2.
Nel nostro caso l'operando destinazione e' maggiore dell'operando
sorgente poiche' il primo e' positivo mentre il secondo e' negativo.
Facciamo un'altro passo e lo verifichiamo:

```
D0: 00009000 00006000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4F64
SSP=07CA6097 USP=07CA4F64 SR=800B T1 -- PL=0 -N-VC PC=07CA7B58
PC=07CA7B58 4E71                NOP
>
```

come potete vedere il salto e' stato effettuato, infatti la prossima istruzione
da eseguire e' la NOP.
Proviamo ora a vedere cosa accade sostituendo alla BGT l'istruzione BHI.
Anche questa istruzione effettua il salto se l'operando destinazione e'
maggiore dell'operando sorgente. La differenza e' che la BHI considera i numeri
tutti positivi.
Eseguiamo il programma modificato.

```
D0: 00009000 00006000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4F64
SSP=07CA6097 USP=07CA4F64 SR=800B T1 -- PL=0 -N-VC PC=07CA7B52
PC=07CA7B52 62000004                BHI.W   $07CA7B58
>
```

Questa volta, $9000 viene considerato come numero positivo. Allora esso
risulta maggiore di $6000. Pertanto il salto non viene effettuato:

```
D0: 00009000 00006000 00000000 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4F64
SSP=07CA6097 USP=07CA4F64 SR=800B T1 -- PL=0 -N-VC PC=07CA7B56
PC=07CA7B56 4E75                RTS
```

```
>
```

In conclusione, quando si usa la CMP bisogna fare molta attenzione a come si vogliono interpretare i numeri negativi, e usare quindi la giusta Bcc.

24.21 Lezionep8

```
; Lezione8p8.s        Funzionamento dei Condition Codes con l'istruzione ADDX

        SECTION        CondC,CODE

Inizio:
        move.l        #$b1114000,d0
        move.l        #$22222222,d1
        move.l        #$82345678,d2
        move.l        #$abababab,d3
        add.l        d0,d2
        addx.l        d1,d3
        move.l        #$01114000,d0
        move.l        #$00000000,d1
        move.l        #$02222222,d2
        move.l        #$00000000,d3
        add.l        d0,d2
        addx.l        d1,d3
stop:
        rts

        end
```

Vediamo ora un esempio d'uso dell'istruzione ADDX.
Supponiamo di dover sommare 2 interi a 64 bit, uno contenuto in D0 e D1
e l'altro in D2 e D3. Per prima cosa sommiamo i 32 bit meno significativi
dei 2 numeri con una normale ADD:

```
D0: B1114000 22222222 82345678 ABABABAB 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8008 T1 -- PL=0 -N--- PC=07CA74C4
PC=07CA74C4 D480                ADD.L   D0,D2
>
```

Notiamo che viene generato un riporto, perche' la somma e' troppo grande per essere contenuta in 32 bit. Pertanto i flag C e X assumono il valore 1.
Per sommare i 32 bit piu' significativi, impieghiamo la ADDX che
aggiunge ai 2 registri anche il contenuto del flag X, tenendo conto cosi'
del riporto.

```
D0: B1114000 22222222 33459678 ABABABAB 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8013 T1 -- PL=0 X--VC PC=07CA74C6
PC=07CA74C6 D781                ADDX.L  D1,D3
>
```

Abbiamo cosi' il nostro risultato a 64 bit nei registri D2 e D3

```
D0: B1114000 22222222 33459678 CDCDCDCE 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8008 T1 -- PL=0 -N--- PC=07CA74C8
PC=07CA7B3E 223C02222222        MOVE.L  #$02222222,D1
>
```

```
La ADDX modifica i flag come la ADD, tranne che per il flag Z.
Infatti il flag Z viene azzerato se il risultato di ADDX e‘ diverso da zero,
ma viene lasciato inalterato se il risultato e‘ zero. Cio‘ consente al flag
Z di tener conto dello stato dell’intera operazione.
Il seguito dell’esempio ce lo mostra:
Ci troviamo infatti a sommare 2 numeri a 64 bit, ma entrambi hanno i 32
bit piu‘ significativi azzerati

D0: 01114000 00000000 02222222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8004 T1 -- PL=0 --Z-- PC=07CA8058
PC=07CA8058 D480                ADD.L   D0,D2
>

La ADD delle cifre meno significative pone Z al valore 1 perche‘ il risultato
non e‘ nullo

D0: 01114000 00000000 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8000 T1 -- PL=0 ----- PC=07CA805A
PC=07CA805A D781                ADDX.L  D1,D3
>

Il risultato della ADDX invece e‘ proprio zero. Se essa si comportasse come
la ADD dovrebbe azzerare il flag Z. Ma anche se la somma dei 32 bit piu‘
significativi e‘ nulla, non lo e‘ il risultato dell’intera operazione.
La ADDX dunque lascia invariato il flag Z in modo tale che noi possiamo
accorgerci che il risultato dell’intera operazione e‘ non nullo

D0: 01114000 00000000 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8000 T1 -- PL=0 ----- PC=07CA805C
PC=07CA805C 4E75                RTS
>

Questa maniera di trattare il flag Z e‘ usato anche dalle istruzioni
SUBX e NEGX.
```

24.22 Lezionep9

```
; Lezione8m9.s        Funzionamento dei Condition Codes con le
;                istruzioni di shift

        SECTION        CondC,CODE

Inizio:
        move.w        #$c003,d0
        move.w        d0,d1
        lsr.w        #1,d0
        asr.w        #1,d1

        move.w        #$6000,d0
        move.w        d0,d1
        lsl.w        #1,d0
        asl.w        #1,d1
stop:
        rts

        end
```

```
In quest'esempio tratteremo le istruzioni di shift, mettendo in luce le
differenze tra le istruzioni di shift aritmetico (ASx) e logico (LSx).
Iniziamo dallo shift a destra. Prendiamo il numero $C003 e lo shiftiamo
a destra di 1 posto (che corrisponde a dividere per 2). Iniziamo
con LSR:

D0: 0000C003 0000C003 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8008 T1 -- PL=0 -N--- PC=07CA78A6
PC=07CA78A6 E248                LSR.W   #1,D0
>

La LSR interpreta i numeri sempre come numeri positivi.
Osserviamo che il numero $C003 e' diventato $6001, il che e' corretto se
lo assumiamo come positivo. Osservate inoltre che il flag C ha assunto il
valore del bit che e' uscito a destra, in questo caso 1.

D0: 00006001 0000C003 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8011 T1 -- PL=0 X---C PC=07CA78A8
PC=07CA78A8 E241                ASR.W   #1,D1
>

La ASR invece interpreta i numeri in complemento a 2. In questo caso, dunque,
$C003 e' stato interpretato come numero negativo, e il risultato ottenuto
e' $E001 che e' corretto nella notazione in complemento a 2 come potete
verificare con il comando "?" di ASMONE.

D0: 00006001 0000E001 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8019 T1 -- PL=0 XN--C PC=07CA78AA
PC=07CA78AA 303C6000                MOVE.W  #$6000,D0
>

Veniamo ora allo shift a sinistra che "corrisponde" alla moltiplicazione.
Anche qui c'e' la stessa differenza tra ASL e LSL. Vediamo prima come si
comporta la LSL:

D0: 00006000 00006000 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8010 T1 -- PL=0 X---- PC=07CA78B0
PC=07CA78B0 E348                LSL.W   #1,D0
>

Come vedete il risultato dello shift a sinistra di $6000 e' $C000 che e'
corretto se interpretiamo $C000 come numero positivo. Vediamo cosa fa
invece la ASL

D0: 0000C000 00006000 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=8008 T1 -- PL=0 -N--- PC=07CA78B2
PC=07CA78B2 E341                ASL.W   #1,D1
>

Il risultato e' ancora $C000. Il che e' sbagliato se interpretiamo i numeri
in complemento a 2. Come mai? Se convertite $6000 in decimale e lo moltiplicate
per 2 vedrete che il risultato e' maggiore di 32767, e pertanto non puo' essere
rappresentato correttamente in notazione complemento a 2. Notate che la ASL
ci segnala il fatto ponendo ad 1 il flag V. Cio' non accade invece con la LSL,
che azzera sempre il flag V. Questa e' la sola (ma importante) differenza
tra le 2 istruzioni di shift a sinistra.
```

```
D0: 0000C000 0000C000 03336222 00000000 00000000 00000000 00000000 00000000
A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 07CA4A64
SSP=07CA5B97 USP=07CA4A64 SR=800A T1 -- PL=0 -N-V- PC=07CA78B4
PC=07CA78B4 4E75                RTS
>
```

Lezionep9b

```
; Lezione8p9b.s                Quesiti e consigli sulle applicazioni dei CC

        SECTION        CondC,CODE

AspettaMouse:
        move.b         $BFE001,d2
        and.b         #$40,D2               ; $40 = %01000000, cioe' bit 6
        bne.s         AspettaMouse
        RTS

        end
```

Come mai questa routine aspetta correttamente la pressione del mouse, senza BTST alcuno? Spero che il commento a lato e la vostra conoscenza dei CC vi faccia supporre la risposta.
Veniamo ad alcune applicazioni dei "cc". Andatevi a ripescare la Lezione7n.s che faceva rimbalzare uno sprite. Ecco quella routine senza i "btst" che testavano (inutilmente) il bit alto per vedere se il numero era divenuto negativo:

```
; Questa routine cambia le coordinate dello sprite aggiungendo una velocita'
; costante sia in verticale che in orizzontale. Inoltre quando lo sprite tocca
; uno dei bordi, la routine provvede a invertire la direzione.
; Per comprendere questa routine occorre sapere che l'istruzione "NEG" serve
; a trasformare un numero positivo in negativo e viceversa. Inoltre noterete
; anche un BPL dopo un ADD, e non dopo un TST o un CMP. Ora sapete perche':


MuoviSprite:
        move.w         sprite_y(PC),d0        ; leggi la vecchia posizione
        add.w         speed_y(PC),d0        ; aggiungi la velocita'
        bpl.s         no_tocca_sopra        ; se >0 va bene
        neg.w         speed_y                ; se <0 abbiamo toccato il bordo superiore
                              ; allora inverti la direzione
        bra.s         Muovisprite        ; ricalcola la nuova posizione

no_tocca_sopra:
        cmp.w         #243,d0        ; quando la posizione vale 256-13=243, lo sprite
                         ; tocca il bordo inferiore
        blo.s         no_tocca_sotto
        neg.w         speed_y               ; se lo sprite tocca il bordo inferiore,
                              ; inverti la velocita'
        bra.s         Muovisprite        ; ricacola la nuova posizione

no_tocca_sotto:
        move         d0,sprite_y        ; aggiorna la posizione
posiz_x:
        move.w         sprite_x(PC),d1        ; leggi la vecchia posizione
        add.w         speed_x(PC),d1        ; aggiungi la velocita'
        bpl.s         no_tocca_sinistra
        neg.w         speed_x                ; se <0 tocca a sinistra: inverti la direzione
        bra.s         posiz_x                ; ricalcola nuova posizione oriz.
```

```
no_tocca_sinistra:
        cmp.w        #304,d1        ; quando la posizione vale 320-16=304, lo sprite
                        ; tocca il bordo destro
        blo.s        no_tocca_destra
        neg.w        speed_x                ; se tocca a destra, inverti la direzione
        bra.s        posiz_x                ; ricalcola nuova posizione oriz.

no_tocca_destra:
        move.w        d1,sprite_x        ; aggiorna la posizione

        lea        miosprite,a1        ; indirizzo sprite
        moveq        #13,d2                ; altezza sprite
        bsr.s        UniMuoviSprite  ; esegue la routine universale che posiziona
                                ; lo sprite
        rts

-        -        -        -        -        -        -        -        -        -

Ora vediamo un'altro possibile utilizzo dei CC. Supponiamo di voler fare uno
scroll verticale ad un bitplane, usando una routine diversa da quella che
"preleva" l'indirizzo dai bplpointers, adda 40 e lo ripunta.
Supponiamo che questa routine debba solo addare 40 al bpl0ptl, ossia alla
word bassa dell'indirizzo. Il problema si pone quando ci troviamo, ad esempio,
all'indirizzo $2ffE2, per cui addando 40 andremmo a $3000a, e anche la word
alta cambia:

Copperlist:
        ...
        dc.w        $e0        ; bpl0pth
PlaneH:
        dc.w        $0002
        dc.w        $e2        ; bpl0ptl
PlaneL:
        dc.w        $ffe2

Come vedete se addiamo 40 a PlaneL otteniamo $000A, ma PlaneH rimane $0002!
Per questo ogni volta preleviamo l'indirizzo, sommiamo e lo rimettiamo nelle
2 word! Altrimenti quando "scatta" la word alta come faremmo?
Con i CC comunque qualcosa si puo' fare. Abbiamo detto che $ffe2+40 ci da
la soluzione esatta, $000a, ma si setta anche il Carry, per il riporto, dato
che abbiamo superato $ffff. Allo potremmo scrivere:

Scroll:
        add.w        #40,PlaneL        ; Scendi di una linea aggiungendo 40 alla
                                ; word bassa dell'indirizzo a cui punta il
                                ; bpl1pt
        bcc.s        NonScattato        ; Abbiamo superato il valore contenibile
                                ; dalla word e dobbiamo modificare anche
                                ; la word alta? Se no salta...
        addq.w        #1,PlaneH        ; Altrimenti adda di 1 la word alta, ossia
                                ; "esegui" il riporto dell'add sul PlaneL!
NonScattato:
        rts

Questi sono alcuni esempi di come si possono "rivedere" routines gia' note.
```

24.23 Lezione8q

```
; Lezione8q.s  Utilizzo delle pic con la palette salvata in fondo (BEHIND).
```

```
;               Tasto sinistro per "colorare", destro per uscire.

        SECTION        Behind,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA

START:
;        puntiamo la figura

        MOVE.L        #Logo1,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP
        MOVEQ        #4-1,D1                ; numero di bitplanes (qua sono 4)
POINTBP:
        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0
        ADD.L        #40*84,d0        ; + lunghezza bitplane (qua e' alto 84 linee)
        addq.w        #8,a1
        dbra        d1,POINTBP


        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA


mouse:
        btst.b        #6,$bfe001        ; Mouse sinistro premuto?
        bne.s        mouse

;                |||||
;        _____.oOo_/o_O\_oOo.____.

Puntacolori:
        lea        logo1+(40*84*4),a0        ; in a0 indirizzo palette dopo pic,
                                        ; ricavabile aggiungendo la lunghezza
                                        ; dei bitplanes all'inizio della
                                        ; pic: rimangono i colori!
        lea        CopColors+2,a1        ; Indirizzo registri colore in coplist
        moveq        #16-1,d0        ; Numero colori
MettiLoop2:
        move.w        (a0)+,(a1)        ; Copia colore da palette a coplist
        addq.w        #4,a1                ; Salta al prossimo registro colore
        dbra        d0,mettiloop2        ; Fai tutti i colori

mouse2:
        btst.b        #2,$dff016        ; Mouse destro premuto?
        bne.s        mouse2
```

```
        rts

*****************************************************************************
;                          Copper List
*****************************************************************************
        section        copper,data_c               ; Chip data

Copperlist:
        dc.w        $8E,$2c81        ; DiwStrt - window start
        dc.w        $90,$2cc1        ; DiwStop - window stop
        dc.w        $92,$38               ; DdfStart - data fetch start
        dc.w        $94,$d0               ; DdfStop - data fetch stop
        dc.w        $102,0               ; BplCon1 - scroll register
        dc.w        $104,0               ; BplCon2 - priority register
        dc.w        $108,0               ; Bpl1Mod - modulo pl. dispari
        dc.w        $10a,0               ; Bpl2Mod - modulo pl. pari

                    ; 5432109876543210
        dc.w        $100,%0100001000000000        ; BPLCON0 - 4 planes lowres (16 colori)

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo         bitplane
        dc.w $ec,$0000,$ee,$0000        ;quarto         bitplane

; i primi 16 colori sono per il LOGO

CopColors:
        dc.w $180,0,$182,0,$184,0,$186,0        ; Ora sono azzerati, sara' la
        dc.w $188,0,$18a,0,$18c,0,$18e,0        ; routine a copiare i valori
        dc.w $190,0,$192,0,$194,0,$196,0        ; dal fondo della pic.
        dc.w $198,0,$19a,0,$19c,0,$19e,0

;        Mettiamo un poco di sfumature per la scenografia...

        dc.w        $8007,$fffe        ; Wait - $2c+84=$80
        dc.w        $100,$200        ; bplcon0 - no bitplanes
        dc.w        $180,$003        ; color0
        dc.w        $8207,$fffe        ; wait
        dc.w        $180,$005        ; color0
        dc.w        $8507,$fffe        ; wait
        dc.w        $180,$007        ; color0
        dc.w        $8a07,$fffe        ; wait
        dc.w        $180,$009        ; color0
        dc.w        $9207,$fffe        ; wait
        dc.w        $180,$00b        ; color0

        dc.w        $9e07,$fffe        ; wait
        dc.w        $180,$999        ; color0
        dc.w        $a007,$fffe        ; wait
        dc.w        $180,$666        ; color0
        dc.w        $a207,$fffe        ; wait
        dc.w        $180,$222        ; color0
        dc.w        $a407,$fffe        ; wait
        dc.w        $180,$001        ; color0

        dc.l        $ffff,$fffe        ; Fine della copperlist
```

```
*****************************************************************************
;                                DISEGNO
*****************************************************************************

        section        gfxstuff,data_c

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).

Logo1:
        incbin        'logo320*84*16c.raw'

        end
```

L'utilita' di mettere la palette nei .raw si nota quando si devono gestire
molte figure, ad esempio in giochi di avventura o negli slideshow.
Ad esempio nel mio "World of Manga" ho usato questo sistema, con le figure
AGA, salvate dall'iffconverter AGA con la palette a 24 bit in fondo.

24.24 Lezione8r

```
; Lezione8r.s                Routines di riconoscimento del processore e del
;                          chipset (aga o normale).
;                          (Ma a noi che ci fa il Sysinfo!!)

        SECTION        SysInfo,CODE

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

               ;5432109876543210
DMASET        EQU        %1000001110000000        ; solo copper e bitplane DMA

START:

;        Puntiamo i bitplanes in copperlist

        MOVE.L        #BITPLANE,d0        ; in d0 mettiamo l'indirizzo del bitplane
        LEA        BPLPOINTERS,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

; NOTATE IL -80!!!! (per provocare l'effetto "profondita'"

        MOVE.L        #BITPLANE-80,d0        ; in d0 mettiamo l'indirizzo del bitplane -80
                                ; ossia una linea SOTTO! *******
        LEA        BPLPOINTERS2,A1        ; puntatori nella COPPERLIST
        move.w        d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap        d0                ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w        d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        bsr.s        CpuDetect        ; Controlla quale CPU e' presente, e cambia
                                ; il testo opportunamente se non e' un 68000
                                ; di base.
        bsr.w        FpuDetect        ; Controlla se e' presente una coprocessore
                                ; matematico in virgola mobile (Floating
                                ; Point Unit).
        bsr.w        AgaDetect        ; Controlla se e' presente il chipset AGA.
```

```
        MOVE.W        #DMASET,$96(a5)               ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA


mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$10800,d2        ; linea da aspettare = $108
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $108
        BNE.S        Waity1
Waity2:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $108
        Beq.S        Waity2

        bsr.w        PrintCarattere        ; Stampa un carattere alla volta

        btst        #6,$bfe001        ; mouse premuto?
        bne.s        mouse

        rts

;***************************************************************************
;                        ROUTINE DI DETECT DEL PROCESSORE
;
; Sia questa routine che quella che controlla la presenza dell'FPU usano un
; apposito byte del sistema operativo, che si trova $129 bytes dopo il
; valore che si trova in $4, ossia execBase+$129.
;
;        AttnFlags ( ossia il byte $129(a6), dove in a6 c'e' execBase )
;
;      bit        CPU o FPU
;
;        0        68010 (o 68020/30/40)
;        1        68020 (o 68030/40)
;        2        68030 (o 68040)                        [V37+]
;        3        68040                                [V37+]
;        4        68881 FPU fitted (o 68882)
;        5        68882 FPU fitted                [V37+]
;        6        68040 FPU fitted                [V37+]
;
;***************************************************************************

;             /\                      ___.                /\
;            /  \   ______  __      /   |_________     /  \NoS!
;           /    \  \_    \/  \    /    |        /    /    \_ ___ _/\__
;          //     \  /         \  /_____|   ____/___./     \ _ _ _ ø¶ /
;         //       \/    \  /   \/       |   \__     |       \\   /_)(_\
;        /          \     \/     \       |     7     |         \    \/
;       /___________\____/_____/\______j___________j___________\

CpuDetect:
        LEA        CpuType(PC),A1
```

```
        move.l        4.w,a6                ; ExecBase in a6

; nota: il 68030/40 non viene riconosciuto dal kickstart 1.3 o inferiore, ma
; si suppone che chi ha un 68020+ ha anche il kickstart 2.0 o superiore!

        btst.b        #3,$129(a6)        ; Attnflags - un 68040?
        BNE.S         M68040
        btst.b        #2,$129(a6);d0        ; Attnflags - un 68030?
        BNE.S         M68030
        btst.b        #1,$129(a6);d0        ; Attnflags - un 68020?
        BNE.S         M68020
        btst.b        #0,$129(a6);d0        ; Attnflags - un 68010?
        BNE.S         M68010
M68000:
        BRA.S         PROCDONE        ; un 68000! lascia la scritta '68000'

M68010:
        MOVE.W        #'10',(a1)        ; cambia '68000' in '68010'
        BRA.S         PROCDONE

M68020:
        MOVE.W        #'20',(a1)        ; cambia '68000' in '68020'
        BRA.S         PROCDONE

M68030:
        MOVE.W        #'30',(a1)        ; cambia '68000' in '68030'
        BRA.S         PROCDONE

M68040:
        MOVE.W        #'40',(a1)        ; cambia '68000' in '68040'


PROCDONE:
        rts

;***************************************************************************
;                        ROUTINE DI DETECT DEL COPROCESSORE
;***************************************************************************

; Ora controllo se e' presente un coprocessore matematico (FPU)

FPUDetect:
        LEA         FpuType(PC),a1        ; stringa di testo del coprocessore (FPU)
        move.l        4.w,a6                ; Execbase (Per accedere al byte AttnFlags)
        btst.b        #3,$129(a6)        ; se e' un 68040, il coprocessore e' incluso!
        BNE.S         FpuPresente
        btst.b        #4,$129(a6);d0        ; 68881? -> FPU detected!
        BNE.S         FpuPresente
        btst.b        #5,$129(a6);d0        ; 68882? -> FPU detected!
        BNE.S         FpuPresente
        BRA.S         FpuNonPresente        ; NO FPU! Vabbe'....

FpuPresente:
        MOVE.L        #'FOUN',(A1)+        ; Se e' presente, scriviamo FOUND!
        MOVE.B        #'D',(A1)+
FpuNonPresente:
        rts

;***************************************************************************
;              ROUTINE DI DETECT DEL CHIPSET AGA (non sbaglia MAI!)
;***************************************************************************
```

```
AgaDetect:
        LEA         $DFF000,A5
        MOVE.W        $7C(A5),D0        ; DeniseID (o LisaID AGA)
        MOVEQ        #100,D7               ; Controlla 100 volte (per sicurezza, dato
                                  ; che il vecchio denise da valori casuali)
DENLOOP:
        MOVE.W        $7C(A5),D1        ; Denise ID (o LisaID AGA)
        CMP.B        d0,d1                ; Lo stesso valore?
        BNE.S        NOTAGA                ; Non e' lo stesso valore: Denise OCS!
        DBRA         D7,DENLOOP
        BTST.L        #2,d0                ; BIT 2 azzerato=AGA. E' presente l'aga??
        BNE.S        NOTAGA                ; no?
        LEA         Chipset(PC),A1        ; SI!
        MOVE.L        #'AGA ',(A1)+        ; Metti AGA al posto di NORMAL...
        MOVE.W        #'  ',(A1)+
        LEA         Messaggio(PC),A1 ; E fai i complimenti per la presenza dell'AGA
        MOVE.L        #'Gran',(A1)+
        MOVE.L        #'de! ',(A1)+
        MOVE.L        #'Una ',(A1)+
        MOVE.L        #'macc',(A1)+
        MOVE.L        #'hina',(A1)+
        MOVE.L        #' AGA',(A1)+
        MOVE.L        #'!!! ',(A1)+
        MOVE.L        #'    ',(A1)+
        MOVE.L        #'    ',(A1)+
        MOVE.L        #'    ',(A1)
NOTAGA:                                ; non AGA... OCS/ECS... mah..
        rts                         ; Allora lascia il messaggio di comprarselo!

*****************************************************************************
;                        Routine di Print
*****************************************************************************

PRINTcarattere:
        MOVE.L        PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        CMP.B        #$ff,d2                ; Segnale di fine testo? ($FF)
        beq.s        FineTesto        ; Se si, esci senza stampare
        TST.B        d2                ; Segnale di fine riga? ($00)
        bne.s        NonFineRiga        ; Se no, non andare a capo

        ADD.L        #80*7,PuntaBITPLANE        ; ANDIAMO A CAPO
        ADDQ.L        #1,PuntaTesto                ; primo carattere riga dopo
                                  ; (saltiamo lo ZERO)
        move.b        (a0)+,d2                ; primo carattere della riga dopo
                                  ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                  ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                  ; DELLO SPAZIO (che e' $20), in $00, quello
                                  ; DELL'ASTERISCO ($21), in $01...
        LSL.W        #3,D2                ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                  ; essendo i caratteri alti 8 pixel
        MOVE.L        D2,A2
        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L        PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                  ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B        (A2)+,(A3)        ; stampa LA LINEA 1 del carattere
```

```
        MOVE.B        (A2)+,80(A3)        ; stampa LA LINEA 2  " "
        MOVE.B        (A2)+,80*2(A3)        ; stampa LA LINEA 3  " "
        MOVE.B        (A2)+,80*3(A3)        ; stampa LA LINEA 4  " "
        MOVE.B        (A2)+,80*4(A3)        ; stampa LA LINEA 5  " "
        MOVE.B        (A2)+,80*5(A3)        ; stampa LA LINEA 6  " "
        MOVE.B        (A2)+,80*6(A3)        ; stampa LA LINEA 7  " "
        MOVE.B        (A2)+,80*7(A3)        ; stampa LA LINEA 8  " "

        ADDQ.L        #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L        #1,PuntaTesto        ; prossimo carattere da stampare

FineTesto:
        RTS


PuntaTesto:
        dc.l        TESTO

PuntaBitplane:
        dc.l        BITPLANE

;        $00 per "fine linea" - $FF per "fine testo"

                ; numero caratteri per linea: 40
TESTO:             ;                  11111111112222222223333333334
            ;   12345678901234567890123456789012345678901234567890
        dc.b        '    Loading Randy Operating System 1.02,'   ; 1
        dc.b        ' please wait...                         ',0 ; 1b
;
        dc.b        '                                        '   ; 2
        dc.b        '                                        ',0 ; 2b
;
        dc.b        '    Testing HARWARE...                  '   ; 3
        dc.b        '                                        ',0 ; 3b
;
        dc.b        '    Testing KickStart...                '   ; 4
        dc.b        '                                        ',0 ; 4b
;
        dc.b        '    Done.                               '   ; 5
        dc.b        '                                        ',0 ; 5b
;
        dc.b        '                                        '   ; 6
        dc.b        '                                        ',0 ; 6b
;
        dc.b        '    PROCESSOR (CPU):  680'
CpuType:
        dc.b        '00             '  ; 7
        dc.b        '                                        ',0 ; 7b
;
        dc.b        '    MATH COPROCESSOR: '
FpuType:
        dc.b        'NONE              '   ; 8
        dc.b        '                                        ',0 ; 8b
;
        dc.b        '    GRAPHIC CHIPSET:  '
Chipset:
        dc.b        'NORMAL           '   ; 9
        dc.b        '                                        ',0 ; 9b
;
        dc.b        '                                        '   ; 10
        dc.b        '                                        ',0 ; 10b
;
```

```
        dc.b        '     '
Messaggio:
        dc.b        'Comprati una macchina AGA!         '   ; 11
        dc.b        '                                    ',$FF ; 11b
;

        EVEN


;        Il FONT caratteri 8x8 copiato in CHIP dalla CPU e non dal blitter,
;        per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin        "assembler2:sorgenti4/nice.fnt"

*****************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $8e,$2c81        ; DiwStrt        (registri con valori normali)
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$003c        ; DdfStart HIRES
        dc.w        $94,$00d4        ; DdfStop HIRES
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2
        dc.w        $108,0               ; Bpl1Mod
        dc.w        $10a,0               ; Bpl2Mod

                    ; 5432109876543210
        dc.w        $100,%1010001000000000        ; bit 13 - 2 bitplanes, 4 colori HIRES

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane
BPLPOINTERS2:
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane

        dc.w        $180,$103        ; color0 - SFONDO
        dc.w        $182,$fff        ; color1 - plane 1 posizione normale, e'
                                ; la parte che "sporge" in alto.
        dc.w        $184,$345        ; color2 - plane 2 (sfasato in basso)
        dc.w        $186,$abc        ; color3 - entrambi i plane - sovrapposizione

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

*****************************************************************************

        SECTION        MIOPLANE,BSS_C        ; Le SECTION BSS devono essere fatte di
                                ; soli ZERI!!! si usa il DS.b per definire
                                ; quanti zeri contenga la section.

;        Ecco perche' serve il "ds.b 80":
;        MOVE.L        #BITPLANE-80,d0        ; in d0 mettiamo l'indirizzo del bitplane -80
;                                ; ossia una linea SOTTO! *******

        ds.b        80        ; la linea che "spunta"
BITPLANE:
        ds.b        80*256        ; un bitplane HIres 640x256

        end
```

Se notate, cambiamo il testo prima che venga stampato, niente di strabiliante.

```
Per sapere quale processore e che chipset ci sono nel computer basta consultare
i relativi bit del sistema operativo e del $dff07c. Comunque fa abbastanza
scena mostrare un detect all'inizio della produzione!!!
```

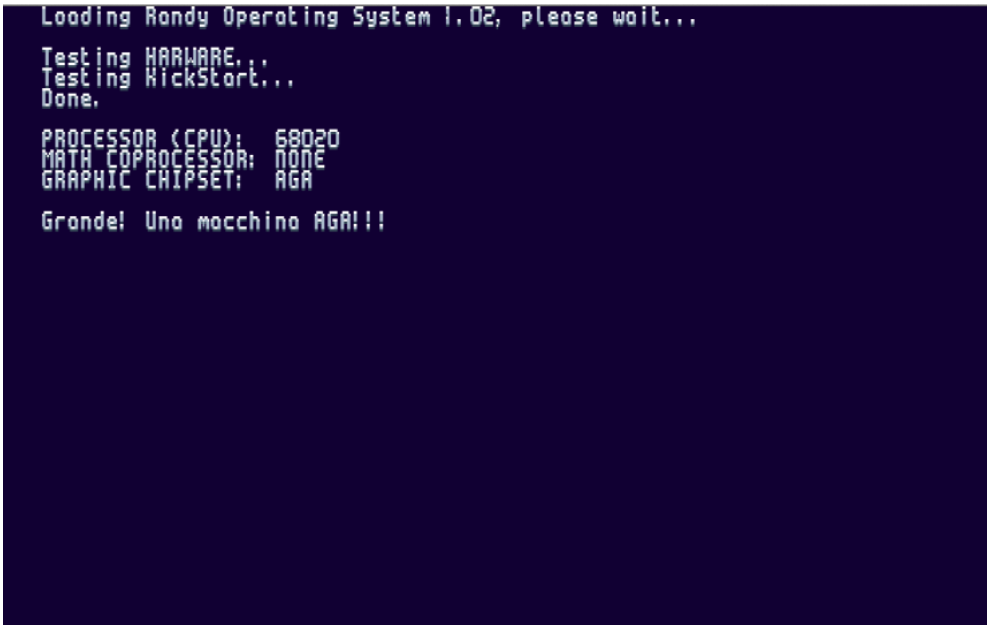


Figura 24.7: Lezione 8r

CAPITOLO 25

LEZIONE 9

25.1 Lezione9a

```
; Lezione9a1.s - AZZERAMENTO DI $10 words tramite il BLITTER
; Prima di vedere questo esempio, date un'occhiata a LEZIONE2f.s dove viene
; cancellata memoria con il 68000

        SECTION Blit,CODE

Inizio:
        move.l  4.w,a6          ; Execbase in a6
        jsr     -$78(a6)        ; Disable - ferma il multitasking
        lea     GfxName,a1      ; Indirizzo del nome della lib da aprire in a1
        jsr     -$198(a6)       ; OpenLibrary
        move.l  d0,a6           ; usa una routine della graphics library:

        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.

                                ; Prima di usare il blitter dobbiamo attendere
                                ; che esso termini eventuali blittate in corso.
                                ; Se ne occupano le istruzioni seguenti

        btst    #6,$dff002      ; attendi che il blitter finisca (test a vuoto)
                                ; per il BUG di Agnus
waitblit:
        btst    #6,$dff002      ; blitter libero?
        bne.s   waitblit

; Ecco come fare una blittata!!! Solo 5 istruzioni per azzerare!!!

;           __
;       __ /_/\   __
;       \/ \_\/  /\_\
;        __   __  \/_/   __
;       /\_\ /\_\  __   /\_\
```

```
;       \/_/ \/_/ /_/\  \/_/
;            __   \_\/
;           /\_\  __
;           \/_/  \/

        move.w  #$0100,$dff040   ; BLTCON0: solo DESTINAZIONE attivata
                                 ; i MINTERMS (cioe' i bits 0-7) sono tutti
                                 ; azzerati. In questo modo si definisce
                                 ; l'operazione di cancellazione

        move.w  #$0000,$dff042   ; BLTCON1: questo registro lo spiegheremo dopo
        move.l  #START,$dff054   ; BLTDPT: Indirizzo del canale di destinazione
        move.w  #$0000,$dff066   ; BLTDMOD: questo registro lo spiegheremo dopo
        move.w  #(1*64)+$10,$dff058 ; BLTSIZE: definisce le dimensioni del
                                    ; rettangolo. In questo caso abbiamo
                                    ; larghezza $10 words e altezza 1 riga.
                                    ; Poiche' l'altezza del rettangolo va
                                    ; scritta nei bit 6-15 di BLTSIZE
                                    ; dobbiamo shiftarla a sinistra di 6 bit.
                                    ; Cio' equivale a moltiplicarne il valore
                                    ; per 64. La larghezza viene espressa nei
                                    ; 6 bit bassi e pertanto non viene
                                    ; modificata.
                                    ; Inoltre questa istruzione da inizio
                                    ; alla blittata

        btst    #6,$dff002      ; attendi che il blitter finisca (test a vuoto)
waitblit2:
        btst    #6,$dff002      ; blitter libero?
        bne.s   waitblit2

        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
        move.l  a6,a1           ; Base della libreria grafica da chiudere
        move.l  4.w,a6
        jsr     -$19e(a6)       ; Closelibrary - chiudo la graphics lib
        jsr     -$7e(a6)        ; Enable - riabilita il Multitasking
        rts

*****************************************************************************

        SECTION THE_DATA,DATA_C

; notate che i dati che cancelliamo devono essere in memoria CHIP
; infatti il Blitter opera solo in memoria CHIP

START:
        dcb.b   $20,$fe
THEEND:
        dc.b    'Qui non cancelliamo'

        even

GfxName:
        dc.b    "graphics.library",0,0

        end
```

Questo esempio e' la versione per blitter del listato Lezione2f.s, in cui si azzeravano dei bytes tramite un loop di "clr.l (a0)+".

Come in quel caso, assemblate, senza Jumpare, e controllate con un "M START"

che sotto tale label sono assemblati $20 bytes "$fe". A questo punto eseguite
il listato, attivando, per la prima volta nel corso, il blitter, dopodiche'
rifate "M START" e verificherete che tali bytes sono stati azzerati, fino alla
label THEEND, infatti con un "N THEEND" troverete la scritta sempre al suo
posto.

L'operazione di cancellazione richiede l'uso del solo canale D.
Inoltre e' necessario azzerare tutti i MINTERMS. Pertanto il valore da caricare
nel registro BLTCON0 e' $0100.
Notate bene il valore che viene scritto nel registro BLTSIZE. Dobbiamo
cancellare un rettangolo largo $10 words e alto una riga. Dobbiamo scrivere
la larghezza nei bit 0-5 di BLTSIZE e l'altezza nei bit 6-15 sempre di BLTSIZE.
Per scrivere l'altezza nei bit 6-15 possiamo quindi shiftarla a sinistra di
6 bit, il che equivale a moltiplicarla per 64. Dunque per scrivere le
dimensioni del rettangolo da blittare nel registro BLTSIZE si usa la seguente
formula:

Valore da scrivere in BLTSIZE = (ALTEZZA*64)+LARGHEZZA

Vi ricordo che la LARGHEZZA e' espressa in words.

NOTA: E' stata usata una funzione del sistema operativo che non abbiamo mai
trattato prima, cioe' quella che impedisce l'uso del blitter al sistema
operativo per evitare di usare il blitter quando anche il workbench lo usa.
Per inibire e riattivare l'uso del blitter da parte del sistema operativo basta
eseguire le apposite routines gia' pronte nel kickstart, piu' in particolare
nella graphics.library: avendo in A6 il GFXBASE, bastera' eseguire un

```
        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
```

Per garantirci che siamo i soli a cercare il blitter, mentre un

```
        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
```

sara' necessario prima di uscire dal programma per riattivare il workbench.

Dunque basta ricordarsi che quando usiamo il blitter nei nostri capolavori e'
necessario aggiungere l'OwnBlitter all'inizio e il DisownBlitter alla fine,
oltre al noto Disable ed Enable.

Lezione9a2

```
; Lezione9a2.s - COPIA DI $10 words tramite il BLITTER

        SECTION Blit,CODE

Inizio:
        move.l  4.w,a6          ; Execbase in a6
        jsr     -$78(a6)        ; Disable - ferma il multitasking
        lea     GfxName,a1      ; Indirizzo del nome della lib da aprire in a1
        jsr     -$198(a6)       ; OpenLibrary
        move.l  d0,a6           ; usa una routine della graphics library:
        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.
        btst    #6,$dff002      ; attendi che il blitter finisca (test a vuoto)
                                ; per il BUG di Agnus
waitblit:
        btst    #6,$dff002      ; blitter libero?
```

```
        bne.s   waitblit

; Ecco come fare una copia

;          __&__
;         /     \
;        |       |
;        |  (o)(o)
;        c   .---_)
;         | |.___|
;         |  \__/
;         /_____\
;        /_____/ \
;       /         \

        move.w  #$09f0,$dff040   ; BLTCON0: attivati canali A e D
                                 ; i MINTERMS (cioe' i bits 0-7) assumono il
                                 ; valore $f0. In questo modo si definisce
                                 ; l'operazione di copia da A a D

        move.w  #$0000,$dff042   ; BLTCON1: questo registro lo spiegheremo dopo
        move.l  #SORG,$dff050    ; BLTAPT: Indirizzo del canale sorgente
        move.l  #DEST,$dff054    ; BLTDPT: Indirizzo del canale di destinazione
        move.w  #$0000,$dff064   ; BLTAMOD: questo registro lo spiegheremo dopo
        move.w  #$0000,$dff066   ; BLTDMOD: questo registro lo spiegheremo dopo
        move.w  #(1*64)+$10,$dff058 ; BLTSIZE: definisce le dimensioni del
                                    ; rettangolo. In questo caso abbiamo
                                    ; larghezza $10 words e altezza 1 riga.
                                    ; Poiche' l'altezza del rettangolo va
                                    ; scritta nei bit 6-15 di BLTSIZE
                                    ; dobbiamo shiftarla a sinistra di 6 bit.
                                    ; Cio' equivale a moltiplicarne il valore
                                    ; per 64. La larghezza viene espressa nei
                                    ; 6 bit bassi e pertanto non viene
                                    ; modificata.
                                    ; Inoltre questa istruzione da inizio
                                    ; alla blittata

        btst    #6,$dff002      ; attendi che il blitter finisca (test a vuoto)
waitblit2:
        btst    #6,$dff002      ; blitter libero?
        bne.s   waitblit2

        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
        move.l  a6,a1           ; Base della libreria grafica da chiudere
        move.l  4.w,a6
        jsr     -$19e(a6)       ; Closelibrary - chiudo la graphics lib
        jsr     -$7e(a6)        ; Enable - riabilita il Multitasking
        rts

GfxName:
        dc.b    "graphics.library",0,0

*****************************************************************************

        SECTION THE_DATA,DATA_C

; notate che i dati che copiamo devono essere in memoria CHIP
; infatti il Blitter opera solo in memoria CHIP

; questa e' la sorgente
```

```
SORG:
        dc.w    $1111,$2222,$3333,$4444,$5555,$6666,$7777,$aaaa
        dc.w    $8888,$2222,$3333,$4444,$5555,$6666,$7777,$ffff
THEEND1:
        dc.b    'Qui finisce la sorgente'
        even

; questa e' la destinazione

DEST:
        dcb.w   $10,$0000
THEEND2:
        dc.b    'Qui finisce la destinazione'

        even

        end
```

Questo esempio mostra una SEMPLICE copia con il blitter.
Assemblate, senza Jumpare, controllate con il comando dell'ASMONE "M SORG"
che a partire dall'indirizzo SORG ci siano in memoria $10 word che assumono
valori vari. Si tratta della sorgente della copia, cioe‘ della zona della quale
leggeremo i dati. Allo stesso modo, con il comando "M DEST" verificate che
a partire dell'indirizzo DEST ci siano $10 word azzerate.

A questo punto eseguite l'esempio.
Ora, sempre con il comando ASMONE "M" andate a vedere quello che e‘ successo
in memoria: i dati all'indirizzo SORG sono rimasti gli stessi di prima.
Cio‘ e‘ normale perche‘ il blitter ha semplicemente letto quei dati, senza
modificarli. Le word a partire dall'indirizzo DEST, invece, ora non sono piu‘
azzerate, ma hanno assunto gli stessi valori dei dati sorgente.

L'operazione di copia richiede l'uso di un canale in lettura e di uno in
scrittura. In questo caso usiamo A in lettura e D (ovviamente) per scrivere.
Per copiare dal canale A al canale D e‘ necessario porre i MINTERMS al valore
$f0. Pertanto il valore da caricare nel registro BLTCON0 e‘ $09f0.

Notate che avremmo potuto eseguire la copia usando un'altro canale (B o C)
per la lettura. Potete provare a farlo voi per esercizio. Le modifiche
da fare sono molto semplici:

- Abilitare il canale che volete usare invece che il canale A (bit 8-11 di
  BLTCON0)

- Cambiare il valore dei MINTERMS (bit 0-7 di BLTCON0) per indicare una copia
  dal canale che volete usare al canale D.
  Per copiare dal canale B al D il valore giusto e‘ $CC, mentre per la copia
  da C a D e‘ $AA.

- Scrivere l'indirizzo di partenza dei dati da copiare invece che nel puntatore
  al canale A (BLTAPT) nel puntatore al canale che volete usare. Gli indirizzi
  dei registri BLTBPT e BLTCPT sono riportati nella lezione.

25.2 Lezione9b1

```
; Lezione9b1.s  BLITTATA, in cui copiamo 8 word in un bitplane azzerato.
;               Tasto sinistro per eseguire la blittata, destro per uscire.

        SECTION CiriCop,CODE
```

```
;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

Aspettasin:
        btst    #6,$bfe001      ; aspetta la pressione del tasto sin. mouse
        bne.s   Aspettasin

        btst.b  #6,2(a5) ; dmaconr
WBlit:
        btst.b  #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit

;        |\/\/\/|
;        |      |
;        |      |
;        | (o)(o)
;        c      _)
;         | ,___|
;         |   /
;        /____\
;       /      \                ; i 2 registri seguenti li spiegheremo in
                                ; seguito:

        move.w  #$ffff,$44(a5)  ; bltafwm - maschera canale A, prima word
        move.w  #$ffff,$46(a5)  ; bltalwm - mask canale A, seconda word

        move.w  #$09f0,$40(a5)  ; bltcon0 - canali A e D abilitati,
                                ; MINTERMS=$f0, ossia copia da A a D
        move.w  #$0000,$42(a5)          ; bltcon1 - lo spiegheremo in seguito
        move.l  #figura_a_caso,$50(a5)  ; bltapt - indirizzo figura sorgente

; l'indirizzo della destinazione dipende dalla posizione X,Y in cui vogliamo
; disegnare il primo pixel della figura. Si applicano le regole della lezione
; In questo caso X=32 e Y=4.

        move.l  #bitplane+(4*20+32/16)*2,$54(a5)        ; bltdpt - ind. dest.
        move.w  #64*1+8,$58(a5)                 ; bltsize - altezza 1 linea,
```

```
                                        ; larghezza 8 words.

mouse:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questa e' la "figura" che viene copiata nel BITPLANE con una blittata:

Figura_a_caso:
        dc.w    $1111,$1010,$2044,$235a
        dc.w    $18f0,$97ff,$ca54,$90a2

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio copiamo una zona di memoria con il blitter.
Piu‘ precisamente leggiamo 8 words (considerandole come un rettangolo largo
8 words e alto una sola linea) a partire dall'indirizzo identificato
dalla label "Figura_a_caso:" e le riscriamo a partire dall'indirizzo
identificato dalla label "BITPLANE:", che, come si capisce dal nome della
label e‘ l'indirizzo di una zona di memoria che contiene un bitplane.
```

In realta' li copiamo in BITPLANE+offset, ossia scostati dall'angolo d'inizio.
Pertanto i dati che copieremo vengono visualizzati sullo schermo.
Per eseguire un operazione di copia e' necessario utilizzare 2 canali
DMA, uno per leggere e uno per scrivere. In questo caso usiamo il canale
A per leggere, e ovviamente il D per scrivere. Pertanto solo questi 2 canali
sono abilitati settando a 1 i relativi bit nel registro BLTCON0.
Per indicare al blitter che deve eseguire una copia dal canale A al canale
D e' necessario porre il byte che contiene i MINTERMS al valore $f0.
Provate a variare la posizione nella quale viene disegnata la figura variando
l'indirizzo destinazione della blittata. Applicate le regole viste nella
lezione.

Lezione9b2

```
; Lezione9b2.s  LOOP DI BLITTATE DI UNA SOLA LINEA

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; in A0 viene memorizzato l'indirizzo della destinazione che varia di volta in
; volta. L'indirizzo iniziale e' calcolato per visualizzare la figura alla
; riga Y=3 a partire dal pixel con X=0

        lea     bitplane+(3*20+0/16)*2,a0       ; indirizzo destinazione
        move.w  #200-1,d7                       ; numero di loop = 200

BlitLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
```

```
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        btst.b  #6,2(a5) ; dmaconr
WBlit:
        btst.b  #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit

;                 (####)
;               (#######)
;             (#########)
;            (#########)
;           (#########)
;          (#########)
;         (#########)
;          (o)(o)(##)
;        ,_c     (##)
;       /____,   (##)
;         \      (#)
;          |    |
;          oooooo
;         /      \

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spiegheremo in seguito
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spiegheremo in seguito
        move.w  #$05CC,$40(a5)          ; BLTCON0 (fa una copia da B a D)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spiegheremo in seguito
        move.w  #$0000,$62(a5)          ; BLTBMOD lo spiegheremo in seguito
        move.w  #$0000,$66(a5)          ; BLTDMOD lo spiegheremo in seguito
        move.l  #figura,$4c(a5)         ; BLTBPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (destinazione variabile a0)
        move.w  #64*1+10,$58(a5)        ; BLTSIZE (via al blitter !)
                                        ; ora, invece di 8 word, come nell
                                        ; esempio precedente, blittiamo 10 word

        add.w   #40,a0                  ; andiamo a blittare alla prossima
                                        ; linea nel prossimo loop.
        dbra    d7,blitloop

mouse:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        btst.b  #6,2(a5) ; dmaconr
WBlit2:
        btst.b  #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
```

```
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************

        SECTION Figura_da_blittare,DATA_C

Figura:
        dc.w    $8888,$aaaa,$cccc,$f0f0
        dc.w    $ffff,$6666,$eeee,$5555
        dc.w    $2222,$dddd

*****************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

*****************************************************************************

Questo esempio e' una variazione dell'esempio lezione9b1.s
Notate come variando l'indirizzo di destinazione i dati vengano copiati
in zone distinte dello schermo. Ogni blittata viene fatta una riga piu' in
basso della precedente. Cio' viene ottenuto aggiungendo sempre 40 (=numero
di bytes per ogni riga) all'indirizzo destinazione.

Notate una cosa molto importante: prima di OGNI blittata si aspetta SEMPRE che
il blitter abbia finito la blittata precedente, mediante il loop Wblit.

In questo esempio abbiamo usato il canale B come canale sorgente.
Conseguentemente, usiamo i registri BLTBPT e BLTBMOD al posto di BLTAPT e
BLTAMOD; inoltre il valore scritto in BLTCON0 e' diverso, perche' dobbiamo
attivare il canale B invece del canale A (quindi il bit 11 vale 0 e il bit 10
vale 1) e perche' dobbiamo porre i MINTERMS al valore $CC che definisce appunto
una copia dal canale B al canale D
```

25.3 Lezione9c1

```
; Lezione9c1.s  PRIMA BLITTATA CON ALTEZZA MAGGIORE DI 1 E MODULI
;               Tasto sinistro per eseguire la blittata, destro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

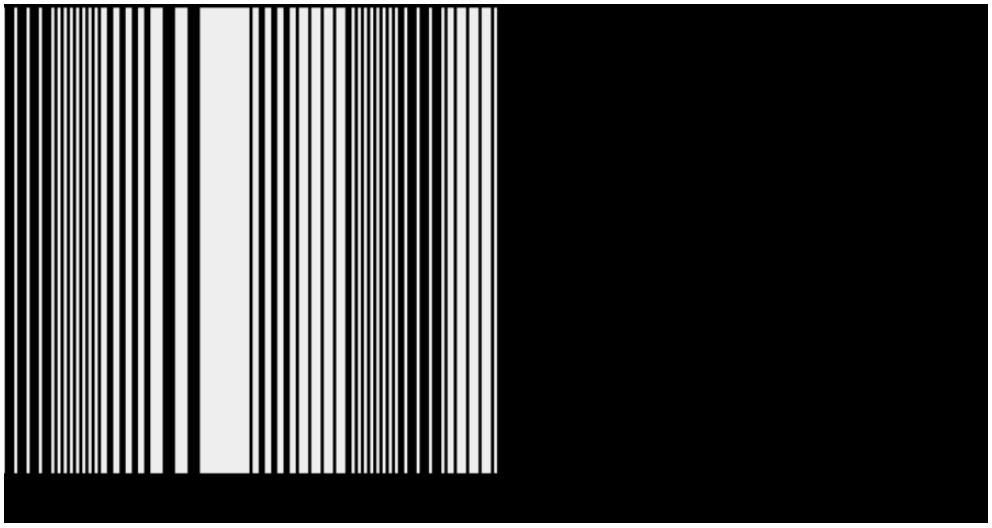


Figura 25.1: Lezione 9b2

```
*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

Aspettasin:
        btst    #6,$bfe001      ; aspetta la pressione del tasto sin. mouse
        bne.s   Aspettasin

        btst.b  #6,2(a5) ; dmaconr
WBlit:
        btst.b  #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit

;           /\  /\
```

```
;       .--/  \/  \---.
;        \           /
;       ._> (o)(o   <__.
;        \  _C        /
;        / /____,  )  \
;       '----\    /----`
;             oooo
;            /    \

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spiegheremo dopo
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spiegheremo dopo
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spiegheremo dopo
        move.w  #0,$64(a5)              ; BLTAMOD =0 perche` il rettangolo
                                        ; sorgente ha le righe consecutive
                                        ; in memoria.

        move.w  #36,$66(a5)             ; BLTDMOD 40-4=36 il rettangolo
                                        ; destinazione e` all'interno di un
                                        ; bitplane largo 20 words, ovvero 40
                                        ; bytes. Il rettangolo blittato
                                        ; e` largo 2 words, cioe` 4 bytes.
                                        ; Il valore del modulo e` dato dalla
                                        ; differenza tra le larghezze

        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  #bitplane,$54(a5)       ; BLTDPT  (linee dello schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; adesso, blitteremo una figura di
                                        ; 2 word X 6 linee con una sola
                                        ; blittata coi moduli opportunamente
                                        ; settati per lo schermo.
mouse:
        btst    #2,$16(a5)      ; tasto destro del mouse premuto?
        bne.s   mouse

        btst.b  #6,2(a5) ; dmaconr
WBlit2:
        btst.b  #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
```

```
        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Definiamo in binario la figura, che e' larga 16 bits, ossia 2 words, e alta
; 6 linee. Notate che le linee sono disposte consecutivamente in memoria.

Figura:
        dc.l    %00000000000000000000110001100000       ; linea 1
        dc.l    %00000000000000000011000110000000       ; linea 2
        dc.l    %00000000000000001100011000000000
        dc.l    %00000110000000110001100000000000
        dc.l    %00000001100011000110000000000000
        dc.l    %00000000011100011000000000000000       ; linea 6

;*****************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

*****************************************************************************

In questo esempio vedete come effettuare la copia di un rettangolo.
Innanzitutto notate ancora una volta la formula da utilizzare per scrivere
le dimensioni del rettangolo in BLTSIZE. Prestate poi molta attenzione a come
sono calcolati i moduli. Per quanto riguarda la sorgente, che inizia alla
label "Figura", le righe del rettangolo sono disposte consecutivamente in
memoria. Pertanto, il valore del modulo per la sorgente (il canale A) vale 0.
Per quanto riguarda la destinazione invece, poiche‘ dobbiamo copiare il
rettangolo all'interno di un bitplane piu‘ largo del rettangolo in questione,
le righe non sono consecutive in memoria e si deve specificare un valore del
modulo calcolato con le formule viste nella lezione.
```

Lezione9c2

```
; Lezione9c2.s          In questo listato una figura di 16*15 pixel, ad
;                       un solo bitplane, viene blittata ripetutamente fino a
;                       riempire lo schermo (320*256 lowres 1 bitplane).

        section bau,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo il primo bitplane
```

```
        MOVE.L  #BitPlane1,d0   ; dove puntare
        LEA     BPLPOINTER1,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.s   fillmem                 ; riempi lo schermo di "mattonelle"
                                        ; col blitter.
mouse:
        btst    #6,$bfe001              ; testa il tasto sin. del mouse
        bne.s   mouse

        rts                             ; uscita


;******************************************************************************
; Questa routine riempie lo schermo di mattonelle.
;******************************************************************************

fillmem:
        lea     Bitplane1,a0    ; indirizzo bitplane destinazione
        lea     gfxdata1,a3     ; fig. mattonella 16*15

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM/LWM - li spiegheremo dopo
        move.w  #0,$64(a5)              ; BLTAMOD = 0, infatti la figura della
                                        ; mattonella NON e' contenuta dentro
                                        ; uno schermo piu' grande e quindi
                                        ; le righe che la compongono sono
                                        ; consecutive in memoria.
        move.w  #38,$66(a5)             ; BLTDMOD (40-2=38), infatti ogni
                                        ; "mattonella" e' larga 16 pixel,
                                        ; cioe' 2 bytes, che dobbiamo togliere
                                        ; alla larghezza totale di una linea,
                                        ; cioe' 40, e il risultato e' 40-2=38!
        move.w  #$0000,$42(a5)          ; BLTCON1 - no modi speciali
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)

        moveq   #16-1,d2                ; 16 file di mattonelle per arrivare
                                        ; verticalmente fino in fondo, infatti
                                        ; le mattonelle sono alte 15 pixel,
                                        ; piu' 1 di "spaziatura" tra una e
                                        ; l'altra, sotto ognuna, fa un
                                        ; ingombro di 16 pixel per piastrella,
                                        ; dunque 256/16=16 mattonelle.
FaiTutteLeRighe:
        moveq   #20-1,d0                ; 20 mattonelle per linea (fila),
                                        ; infatti, essendo le mattonelle
                                        ; larghe 16 pixel, cioe' 2 bytes, ne
```

```
                                    ; deriva che ce ne possono stare
                                    ; 320/16=20 per linea orizzontale.
FaiUnaRigaLoop:
        move.l  a0,$54(a5)          ; BLTDPT - destinazione (bitpl 1)
        move.l  a3,$50(a5)          ; BLTAPT - sorgente (fig1)
        move.w  #(15*64)+1,$58(a5)  ; BLTSIZE - altezza 15 words,
                                    ;           larghezza 1 word (16 pix.)
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        addq.w  #2,a0   ; salta 1 word (16 pixel) nel bitplane 1, scattando
                        ; in "avanti" per la prossima mattonella

        dbra    d0,FaiUnaRigaLoop       ; e cicla fino a che non sono state
                                        ; blittate tutte le 20 mattonelle
                                        ; di una riga.

        lea     15*40(a0),a0    ; salta 15 linee nel bitplane 1. Siccome
                                ; abbiamo gia' incrementato a0 a forza di
                                ; addq #2,a0, abbiamo gia' saltato una linea
                                ; intera prima di arrivare qua. Per ogni loop,
                                ; dunque, vengono saltate 16 linee, lasciando
                                ; tra una mattonella e l'altra una "striscia"
                                ; di sfondo azzerato, dato che le mattonelle
                                ; sono alte solo 15 pixel.
        dbra    d2,FaiTutteLeRighe      ; fai tutte le 16 righe
        rts

;******************************************************************************

                section cop,data_C

copperlist
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w $100,$1200         ; BPLCON0 - 1 bitplane lowres

        dc.w $180,$126  ; Color0
        dc.w $182,$0a0  ; Color1

BPLPOINTER1:
        dc.w $e0,0,$e2,0        ;primo   bitplane

        dc.l    $ffff,$fffe     ; fine della copperlist

;******************************************************************************

;       Figura, composta da 1 biplane. larghezza = 1 word, altezza = 15 linee

gfxdata1:
        dc.w    %1111111111111100       ; 1
        dc.w    %1111111111111100       ; 2
        dc.w    %1100000000001100       ; 3
```

```
        dc.w    %1100000000001100
        dc.w    %1100011110001100
        dc.w    %1100111111001100
        dc.w    %1100110011001100
        dc.w    %1100110011001100
        dc.w    %1100111111001100
        dc.w    %1100011110001100
        dc.w    %1100000000001100
        dc.w    %1100000000001100
        dc.w    %1111111111111100
        dc.w    %1111111111111100
        dc.w    %0000000000000000       ; 15

        section gnippi,bss_C

bitplane1:
                ds.b    40*256

        end

;***************************************************************************
```

In questo esempio usiamo una piccola figura (larga 16 pixel e alta 15 linee) come "mattonella" per "piastrellare" lo schermo. In pratica copiamo la figura sorgente tante volte, in modo da ricoprire tutto lo schermo. Poiche‘ lo schermo e‘ largo 320 pixel e la mattonella 16, in una riga disegnamo 320/16=20 mattonelle. In altezza invece lo schermo misura 256 pixel e la mattonella 15. Siccome lasciamo una riga di pixel vuota tra 2 file di mattonelle, 256/(15+1)=16 mattonelle per ogni colonna.
Ogni mattonella viene copiata mediante una blittata. Le dimensioni della blittata sono di 1 word (16 pixel) in larghezza e 15 righe in altezza.
Il modulo della sorgente vale 0, perche‘ la sorgente NON appartiene ad uno schermo, e le righe che compongono la figura della mattonella sono disposte consecutivamente in memoria. La destinazione invece e‘ dentro uno schermo largo 20 words, e quindi il modulo viene calcolato secondo la fomula vista nella lezione.
Le istruzioni che eseguono la blittata si trovano all’interno di 2 loop posti uno dentro l’altro. Il loop piu‘ interno ripete la blittata 20 volte, in modo da disegnare una fila orizzontale di mattonelle. Il loop piu‘ esterno fa ripetere il loop interno 16 volte, in modo da disegnare in totale 16 file di mattonelle. Tra una blittata e l’altra varia naturalmente l’indirizzo della destinazione in modo da disegnare la mattonella ogni volta in un punto diverso dello schermo. Per questo motivo metteremo il puntatore alla destinazione in un registro che modificheremo durante la routine.
Nel loop interno, disegnamo una alla volta, le mattonelle che formano una fila orizzontale. Quindi dopo aver disegnato una mattonella, dobbiamo spostare il puntatore alla destinazione di una word verso destra, cioe‘ dobbiamo farlo puntare alla word seguente in memoria. Cio‘ equivale ad aggiungere 2 all’indirizzo (una word = 2 bytes). In questo modo quando arriviamo all’ultima iterazione del ciclo interno, il puntatore alla destinazione punta all’ultima word della riga. Dopo la stampa della mattonella (che e‘ l’ultima della fila orizzontale) viene aggiunto ancora 2 al puntatore, facendolo puntare alla prima word della riga seguente. Noi invece vogliamo iniziare a stampare un’altra fila di mattonelle. Siccome una fila di mattonelle e‘ alta 16 linee, dobbiamo disegnare la prossima fila 16 linee piu‘ in basso di quella che abbiamo appena terminato. Il nostro puntatore invece come abbiamo detto punta una linea piu‘ in basso di quella attuale. Quindi, dobbiamo farlo puntare altre 15 linee piu‘ in basso. Cio‘ equivale ad aggiungere 15*40 all’indirizzo, perche‘ ogni riga occupa 40 bytes (20 words), cosa che viene fatta ad ogni iterazione del ciclo esterno.

```
                prima di iniziare la prima iterazione del ciclo interno
                il puntatore punta qui.

                   |
                   V

riga Y          |      |      |      |
riga Y+1        |      |      |      |
.
.                  ^
                   |

                dopo l'ultima iterazione del ciclo interno
                il puntatore punta a questa word.

                Per stampare la nuova fila invece deve puntare a QUESTA word
                Per farcelo arrivare dobbiamo spostarlo in basso di 15 linee
                aggiungendogli 40 per ogni linea.

                   |
                   V

riga Y+16       |      |      |      |
```

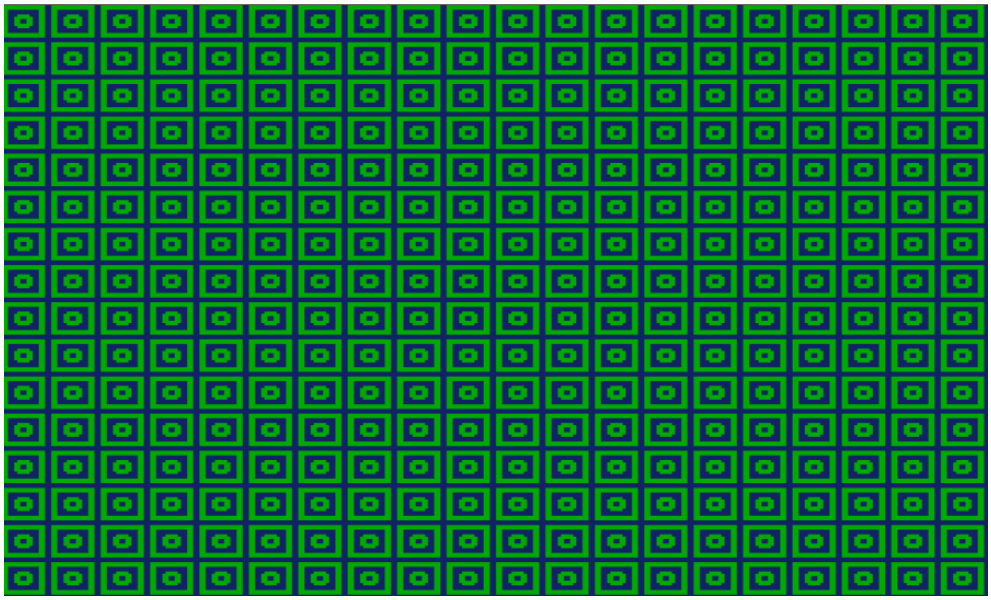


Figura 25.2: Lezione 9c2

Lezione9c3

```
; Lezione9c3.s  BLITTATA con modulo negativo.
;               Tasto sinistro per eseguire la blittata, destro per uscire.

        SECTION CiriCop,CODE
```

```
;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

Aspettasin:
        btst    #6,$bfe001      ; aspetta la pressione del tasto sin. mouse
        bne.s   Aspettasin

        btst    #6,2(a5) ; dmaconr
WBlit:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit
;             __
;            /\ \
;           /  \ \
;          / /\ \ \
;         / / /\ \ \
;        / / /__\_\ \
;       / / /________\
;       \/__________/           ; i 2 registri seguenti li spiegheremo in
                                ; seguito:

        move.w  #$ffff,$44(a5)  ; bltafwm - maschera canale A, prima word
        move.w  #$ffff,$46(a5)  ; bltalwm - mask canale A, seconda word

        move.w  #$09f0,$40(a5)  ; bltcon0 - canali A e D abilitati,
                                ; MINTERMS=$f0, ossia copia da A a D

        move.w  #$0000,$42(a5)          ; bltcon1 - lo spiegheremo in seguito

        move.w  #2*(20-8),$66(a5)       ; BLTDMOD - come al solito.

        move.w  #-16,$64(a5)            ; BLTAMOD - la figura e' larga 8 words
                                        ; (16 bytes): per tornare all'inizio
                                        ; mettiamo il modulo negativo.

        move.l  #figura_a_caso,$50(a5)  ; bltapt - indirizzo figura sorgente
```

```
; L'indirizzo della destinazione dipende dalla posizione X,Y in cui vogliamo
; disegnare il primo pixel della figura. Si applicano le regole della lezione
; In questo caso X=32 e Y=4.

        move.l  #bitplane+(4*20+32/16)*2,$54(a5)        ; bltdpt - ind. dest.
        move.w  #64*10+8,$58(a5)                ; bltsize - altezza 10 linee,
                                                ; larghezza 8 words.

mouse:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questa e' la "figura" che viene copiata nel BITPLANE con una blittata:

Figura_a_caso:
        dc.w    $1111,$1010,$2044,$235a
        dc.w    $18f0,$97ff,$ca54,$90a2


        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio abbiamo una figura alta una sola linea che dobbiamo copiare
a partire da una certa riga dello schermo, 10 volte, ogni volta scendendo
di una riga. Naturalmente potremmo semplicemente fare un loop di 10 blittate,
modificando ogni volta l'indirizzo destinazione. E' possibile pero' farlo con
una sola blittata, mettendo un valore negativo al modulo della sorgente.
Come sapete, il valore del modulo viene aggiunto all'indirizzo contenuto
nel registro puntatore, ogni volta che il blitter finisce di blittare una riga.
Normalmente nel modulo si mette un valore positivo che permette al blitter
di "saltare" le word che non appartengono al rettangolo, andando alla riga
successiva. Se pero' il modulo ha un valore negativo, fara' "tornare indietro"
l'indirizzo contenuto nel registro puntatore. In particolare se la blittata
e' larga L words, blittando una riga il valore contenuto nel puntatore
aumentera' di 2*L (perche' il puntatore conta i bytes, e 1 word = 2 bytes).
Se mettiamo nel modulo il valore -2*L, faremo ritornare il puntatore
esattamente all'inizio della riga. In questo esempio facciamo proprio questo
con la sorgente, rileggendo ogni volta la stessa linea. Per la destinazione
invece, ci comportiamo normalmente, e quindi le 10 linee vengono scritte
una sotto l'altra.
Se vi ricordate abbiamo fatto un effetto simile con i moduli dei bitplanes,
mettendoli a -40, ottenendo un "allungamento" infinito della prima linea,
ma solo a livello di visualizzazione. In questo caso col blitter invece e'
proprio in MEMORIA che riscriviamo la stessa linea varie volte.

25.4 Lezione9d1

```
; Lezione9d1.s  LOOP DI UNA BLITTATA CON ALTEZZA 6 LINEE E MODULI


        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     bitplane,a0     ; indirizzo bitplane destinazione
        move.w  #(150-6)-1,d7   ; -6 perche' la figura e' alta 6 linee,
                                ; percui "arriva" 6 linee piu' in basso di
```

```
                                ; dove si blitta.
BlitLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        btst    #6,2(a5) ; dmaconr
WBlit:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit

;       /\ ___
;       \ (_ _)
;        \ \ª/ )
;         \ , /
;         / \/\
;        /__I__\

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spieghiamo in seguito
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spieghiamo in seguito
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo in seguito
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #36,$66(a5)             ; BLTDMOD (40-4=36)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (dest: linee di schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; adesso, blitteremo una figura di
                                        ; 2 word X 6 linee con una sola
                                        ; blittata dai moduli opportunamente
                                        ; settati correttamente per lo schermo.

        add.w   #40,a0                  ; andiamo a blittare alla prossima
                                        ; linea nel prossimo loop.
                                        ; 40 e‘ il numero di bytes in una riga.
                                        ; aggiungendo tale numero, ci spostiamo
                                        ; in basso di una riga.

        dbra    d7,blitloop

mouse:

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        rts

;******************************************************************************
```

```
        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Definiamo in binario la figura, che e' larga 16 bits, ossia 2 words, e alta
; 6 linee

Figura:
        dc.l    %00000000000000000000110001100000
        dc.l    %00000000000000000011000110000000
        dc.l    %00000000000000001100011000000000
        dc.l    %00000110000000110001100000000000
        dc.l    %00000001100011000110000000000000
        dc.l    %00000000011100011000000000000000

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio vediamo un esempio di animazione con il blitter.
Molto semplicemente  si disegna con il blitter la figura in una posizione
sempre piu' bassa ad ogni vertical blank.
La posizione della figura viene determinata dall'indirizzo che viene scritto
nel registro BLTDPT, notate infatti che il valore scritto in questo registro
viene cambiato ad ogni frame:

        add.w   #40,a0                  ; andiamo a blittare alla prossima
                                        ; linea nel prossimo loop.
                                        ; 40 e' il numero di bytes in una riga.
                                        ; aggiungendo tale numero, ci spostiamo
                                        ; in basso di una riga.

La figura "lascia la scia", perche' non cancelliamo ogni volta la figura che
abbiamo blittato prima. E' come un brush (pennello) del DPaint.
```

Lezione9d2

```
; Lezione9d2.s  LOOP DI UNA BLITTATA CON ALTEZZA 6 LINEE E MODULI, in cui
;               azzeriamo anche tutto lo schermo ogni volta per evitare
;               che rimanga la "scia".


        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     bitplane,a0     ; indirizzo bitplane destinazione
        move.w  #(150-6)-1,d7   ; -6 perche' la figura e' alta 6 linee,
                                ; percui "arriva" 6 linee piu' in basso di
                                ; dove si blitta.
BlitLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

;       Cancellazione dello schermo
```

```
        move.w  #$0100,$40(a5)          ; BLTCON0 - accende solo il canale D,
                                        ; questo provoca la cancellazione della
                                        ; DESTINAZIONE, dato che non c'e' la
                                        ; sorgente!!!
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo in seguito
        move.w  #$0000,$66(a5)          ; BLTDMOD = 0, infatti le righe
                                        ; del bitplane sono disposte
                                        ; consecutivamente in memoria

        move.l  #bitplane,$54(a5)       ; BLTDPT - destinazione (bitplane)
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE - alt.256 linee, largh. 20 w.
                                        ; cancella TUTTO LO SCHERMO, infatti
                                        ; le linee sono 256, (64*256) e i
                                        ; byte per linea sono 40 (20 words)

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

;       Copia della figura

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spieghiamo in seguito
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spieghiamo in seguito
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo in seguito
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #36,$66(a5)             ; BLTDMOD (40-4=36)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (dest: linee di schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; adesso, blitteremo una figura di
                                        ; 2 word X 6 linee con una sola
                                        ; blittata dai moduli opportunamente
                                        ; settati correttamente per lo schermo.

        add.w   #40,a0                  ; andiamo a blittare alla prossima
                                        ; linea nel prossimo loop.
        dbra    d7,blitloop

mouse:

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
```

```
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Definiamo in binario la figura, che e' larga 16 bits, ossia 2 words, e alta
; 6 linee

Figura:
        dc.l    %00000000000000000000110001100000
        dc.l    %00000000000000000011000110000000
        dc.l    %00000000000000001100011000000000
        dc.l    %00000110000000110001100000000000
        dc.l    %00000001100011000110000000000000
        dc.l    %00000000111000110000000000000000

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio miglioriamo il precedente lezione9d1.s.
La figura ora non lascia piu' la scia, perche' cancelliamo ogni volta l'intero
schermo. In realta' e' un po' troppo cancellare tutto lo schermo, basterebbe
cancellare il rettangolino interessato. Comunque finziona!
```

Lezione9d3

```
; Lezione9d3.s  BLITTA VERSO DESTRA, a scatti di 1 word (senza usare shift)


        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

Figura 25.3: Lezione 9d2

```
START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     bitplane,a0             ; destinazione
        moveq   #50-1,d7                ; Num. di spostamenti a destra
MoveLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2
```

```
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

;       Cancellazione dello schermo

        move.w  #$0100,$40(a5)          ; BLTCON0 - accende solo il canale D,
                                        ; questo provoca la cancellazione della
                                        ; DESTINAZIONE, dato che non c'e' la
                                        ; sorgente!!!
        move.w  #$0000,$42(a5)          ; BLTCON1 - lo spieghiamo dopo
        move.w  #$0000,$66(a5)          ; BLTDMOD = 0
        move.l  #bitplane,$54(a5)       ; BLTDPT - destinazione = bitplane
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE - alt.256 linee, largh. 20 w.
                                        ; cancella TUTTO LO SCHERMO, infatti
                                        ; le linee sono 256, (64*256) e i
                                        ; byte per linea sono 40 (20 words)

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spieghiamo dopo
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spieghiamo dopo
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo dopo
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #36,$66(a5)             ; BLTDMOD (40-4=36)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; adesso, blitteremo una figura di
                                        ; 2 word X 6 linee con una sola
                                        ; blittata dai moduli opportunamente
                                        ; settati correttamente per lo schermo.

        addq.w  #2,a0                   ; modifica l'indirizzo, puntando alla
                                        ; word seguente per la prossima
                                        ; blittata. La figura scatta in avanti
                                        ; di 16 pixel
        dbra    d7,moveloop

mouse:

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
```

```
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $100,$1200      ; bplcon0 - 1 bitplane Lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Definiamo in binario la figura, che e' larga 16 bits, ossia 2 words, e alta
; 6 linee

Figura:
        dc.l    %00000000000000000000110001100000
        dc.l    %00000000000000000011000110000000
        dc.l    %00000000000000001100011000000000
        dc.l    %00000110000000110001100000000000
        dc.l    %00000001100011000110000000000000
        dc.l    %00000000011100011000000000000000

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

Questo esempio e‘ analogo a lezione9d2.s, solo che ci spostiamo a destra invece
che in basso. Lo spostamento avviane modificando l'indirizzo destinazione
sul quale blittiamo, spostandoci di una word alla volta. Cio‘ equivale
a spostarci di 16 pixel alla volta (bleah!).
```

25.5 Lezione9e1

```
; Lezione9e1.s          * SHIFTAMENTO * del blitter.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

```
START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        moveq   #0,d4                   ; coordinata orizzontale a 0

Loop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        move.w  d4,d5   ; coordinata orizzontale attuale in d5

        and.w   #$000f,d5       ; Si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d5           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d5           ; della word... (8+4 = shift di 12 bit!)
        or.w    #$09f0,d5       ; ...giusti per inserirsi nel registro BLTCON0
                                ; Qua mettiamo $f0 nei minterm per copia da
                                ; sorgente A a destinazione D e abilitiamo
                                ; ovviamente i canali A+D con $0900 (bit 8
                                ; per D e 11 per A). Ossia $09f0 + shift.

        addq.w  #1,d4           ; Aggiungi 1 alla coordinata orizzontale per
                                ; andare a destra di 1 pixel la prossima volta

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spieghiamo dopo
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spieghiamo dopo
        move.w  d5,$40(a5)              ; BLTCON0 (usa A+D) - nel registro
                                        ; abbiamo messo lo shift! (bits 12,13
                                        ; 14 e 15, ossia nibble alto!)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo dopo
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
```

```
        move.w  #38,$66(a5)             ; BLTDMOD (40-2=38)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  #bitplane,$54(a5)       ; BLTDPT  (linee di schermo)
        move.w  #(64*6)+1,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; la figura e' larga 1 word e alta
                                        ; 6 linee
        btst    #6,$bfe001              ; mouse premuto?
        bne.s   loop

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Ecco il pesce... largo 16 pixel (1 word) e alto 6 linee.

Figura:
        dc.w    %1000001111100000
        dc.w    %1100111111111000
        dc.w    %1111111111101100
        dc.w    %1111111111111110
        dc.w    %1100111111111000
        dc.w    %1000001111100000

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

```
In questo esempio potete vedere come opera lo shift. Abbiamo una figura larga
1 word e alta 6 linee.
Questa figura viene blittata sempre allo stesso indirizzo destinazione, ossia
viene messo sempre lo stesso indirizzo in BLTDPT ($dff054).
Ogni volta, pero‘ viene aumentato di 1 il valore di shift nel BLTCON0.
In questo modo la figura si sposta ogni volta di 1 pixel a destra.
Notate bene il fenomeno descritto nella lezione: i bit che vengono shiftati
fuori dalla word rientrano a sinistra nella word successiva.
Nel nostro caso questo non va bene, perche‘ il muso del pesce esce a destra e
rientra a sinistra, dietro alla coda.
Nel prossimo esempio vedremo come risolvere il problema.
```

Lezione9e2

```
; Lezione9e2.s          SHIFTAMENTO CON oggetto largo 2 word (una azzerata)

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        moveq   #0,d4                   ; coordinata orizzontale a 0

Loop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
```

```
        Beq.S   Waity2

        move.w  d4,d5   ; coordinata orizzontale attuale in d5

        and.w   #$000f,d5       ; Si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d5           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d5           ; della word... (8+4 = shift di 12 bit!)
        or.w    #$09f0,d5       ; ...giusti per inserirsi nel registro BLTCON0
                                ; Qua mettiamo $f0 nei minterm per copia da
                                ; sorgente A a destinazione D e abilitiamo
                                ; ovviamente i canali A+D con $0900 (bit 8
                                ; per D e 11 per A). Ossia $09f0 + shift.

        addq.w  #1,d4           ; Aggiungi 1 alla coordinata orizzontale per
                                ; andare a destra di 1 pixel la prossima volta

        move.w  #$ffff,$44(a5)          ; BLTAFWM lo spieghiamo dopo
        move.w  #$ffff,$46(a5)          ; BLTALWM lo spieghiamo dopo
        move.w  d5,$40(a5)              ; BLTCON0 (usa A+D) - nel registro
                                        ; abbiamo messo lo shift! (bits 12,13
                                        ; 14 e 15, ossia nibble alto!)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo dopo
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #36,$66(a5)             ; BLTDMOD (40-4=36)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  #bitplane,$54(a5)       ; BLTDPT  (linee di schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; blittiamo 2 word, la seconda delle
                                        ; quali e' nulla per permettere
                                        ; lo shift
        btst    #6,$bfe001              ; mouse premuto?
        bne.s   loop

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        rts

;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
```

```
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; ecco il pesce... questa volta abbiamo una seconda word azzerata per ogni riga
; dimensioni: 32*6

Figura:
        dc.w    %1000001111100000,%000000000000000
        dc.w    %1100111111111000,%000000000000000
        dc.w    %1111111111101100,%000000000000000
        dc.w    %1111111111111110,%000000000000000
        dc.w    %1100111111111000,%000000000000000
        dc.w    %1000001111100000,%000000000000000

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempo spostiamo una figura verso destra di un pixel alla volta
con lo shift, utilizzando una word azzerata a destra di ogni riga per
milgiorare l'effetto rispetto a Lezione9e1.s
Poiche' non incrementiamo mai l'indirizzo destinazione, la figura si sposta
solo usando lo shifter del blitter.
In questo modo e' possibile spostarsi al massimo di 15 pixel, poiche' 15 e' il
massimo valore di shift consentito.
Dopo aver raggiunto 15, il valore di shift (che e' ottenuto prendendo i 4 bit
meno significativi della posizione della figura) ritorna al valore 0 e pertanto
la figura ritornera' alla posizione di partenza per ricominciare a muoversi.
Per fare uno scroll "serio", ogni 15 pixel di scorrimento ottenuto con lo
shift occorrerebbe far scattare l'immagine di 16 pixel aggiungendo 2 alla
destinazione, e ripartendo con lo shift a zero, eccetera, in modo analogo a
quanto visto per lo scroll con bplcon1 ($dff102) e bplpointers in copperlist.
```

Lezione9e3

```
; Lezione9e3.s  Spostamento orizzontale completo con shift + cambiamento di
;               posizione della destinazione (scatti di 2 bytes = 16 pixel)

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

Figura 25.4: Lezione 9e2

```
START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        moveq   #0,d1                   ; Coordinata orizzontale a 0
        move.w  #(320-32)-1,d7          ; Muove per 320 pixel MENO la larghezza
                                        ; reale del BOB, per fare in modo che
                                        ; il suo primo pixel a sinistra si
                                        ; fermi quando quello a destra arriva
                                        ; alla fine dello schermo.
Loop:
        cmp.b   #$ff,$6(a5)     ; VHPOSR - aspetta la linea $ff
        bne.s   loop
Aspetta:
        cmp.b   #$ff,$6(a5)     ; ancora linea $ff?
        beq.s   Aspetta

        lea     bitplane,a0     ; destinazione in a0
        move.w  d1,d0
        and.w   #$000f,d0       ; Si selezionano i primi 4 bit perche' vanno
```

```
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; I 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word... (8+4= shift di 12 bit)
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
                                ; Qua mettiamo $f0 nei minterm per copia da
                                ; sorgente A a destinazione D e abilitiamo
                                ; ovviamente i canali A+D con $0900 (bit 8
                                ; per D e 11 per A). Ossia $09f0 + shift.
        move.w  d1,d2
        lsr.w   #3,d2           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d2       ; escludo il bit 0
        add.w   d2,a0           ; Somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione
        addq.w  #1,d1           ; Aggiungi 1 alla coordinata orizzontale

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

; ora, come spiegato nella teoria, cogliamo l' occasione per apportare una
; modifica: scriviamo i valori in registri ADIACENTI con un singolo 'move.l'

        move.l  #$01000000,$40(a5)      ; BLTCON0 + BLTCON1
        move.w  #$0000,$66(a5)
        move.l  #bitplane,$54(a5)
        move.w  #(64*256)+20,$58(a5)    ; provate a togliere questa linea
                                        ; e lo schermo non verra' pulito,
                                        ; dunque il pesce lascera' la "scia"

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo
        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000024,$64(a5)      ; BLTAMOD (=0) + BLTDMOD (=40-4=36=$24)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; blittiamo 2 word, la seconda delle
                                        ; quali e' nulla per permettere
                                        ; lo shift

        btst    #6,$bfe001              ; mouse premuto?
        beq.s   quit

        dbra    d7,loop

Quit:
        rts

;******************************************************************************

        SECTION GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; BplCon0 - 1 bitplane LowRes

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Il pesciolino:

Figura:
        dc.w    %1000001111100000,0
        dc.w    %1100111111111000,0
        dc.w    %1111111111101100,0
        dc.w    %1111111111111110,0
        dc.w    %1100111111111000,0
        dc.w    %1000001111100000,0

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio spostiamo la nostra figura di un numero arbitrario di pixel.
La coordinata orizzontale della figura e‘ memorizzata in D1. Tale coordinata
viene divisa per 8 in modo da calcolare l’indirizzo di memoria della word
a cui essa appartiene. I 4 bit meno significativi della coordianta, invece,
sono il valore di shift, come spiegato nella lezione.
```

25.6 Lezione9f1

```
; Lezione9f1.s  BLITTATA, in cui copiamo un rettangolo da un punto
;               all’altro dello stesso schermo
;               Tasto sinistro per eseguire la blittata, destro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
```

```
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, non cancellare

        bsr.s   copia           ; esegui la routine di copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts

; ********************** LA ROUTINE DI COPIA ***************************

 ; viene copiato un rettangolo con larghezza=160 e altezza=20
 ; dalle coordinate X1=64, Y1=50 (sorgente)
 ; alle coordinate X2=80, Y2=190 (destinazione)

copia:

; Carica gli indirizzi sorgente e destinazione in 2 variabili

        move.l  #bitplane+((20*50)+64/16)*2,d0  ; indirizzo sorgente
        move.l  #bitplane+((20*190)+80/16)*2,d2 ; indirizzo destinazione

                                ; Loop di blittate
        moveq   #3-1,d1         ; ripeti per tutti i bit-planes
copia_loop:
        btst    #6,2(a5)        ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit
```

```
        move.l  #$09f00000,$40(a5)      ; bltcon0 e BLTCON1 - copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo

; carica i puntatori

        move.l  d0,$50(a5)      ; bltapt
        move.l  d2,$54(a5)      ; bltdpt

; Queste 2 istruzioni settano i moduli della sorgente e della destinazione
; notate che poiche‘ sorgente e destinazione sono all’interno dello stesso
; schermo il modulo e‘ lo stesso.
; il modulo e‘ calcolato secondo la formula (H-L)*2  (H e‘ la larghezza del
; bitplane in words e L e‘ la larghezza dell’immagine, sempre in words)
; che abbiamo visto a lezione, (20-160/16)*2=20

        move.w  #(20-160/16)*2,$64(a5)  ; bltamod
        move.w  #(20-160/16)*2,$66(a5)  ; bltdmod

; Notate anche che poiche‘ i 2 registri hanno indirizzi consecutivi, si puo‘
; usare una sola istruzione invece che 2 (ricordate che 20=$14):
; move.l #$00140014,$64(a5)     ; bltamod e bltdmod

        move.w  #(20*64)+160/16,$58(a5)         ; bltsize
                                                ; altezza 20 linee
                                                ; largo 160 pixel (= 10 words)

; Aggiorna le variabili contenenti gli indirizzi per farle puntare
; ai bitplanes seguenti

        add.l   #40*256,d2      ; indirizzo destinazione prossimo plane
        add.l   #40*256,d0      ; indirizzo sorgente prossimo plane

        dbra    d1,copia_loop

        btst    #6,$02(a5)      ; aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$3200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
```

```
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.raw"        ; qua carichiamo la figura

        end

;*****************************************************************************
```

In questo esempio copiamo con il blitter un'immagine formata da 3 bitplanes.
Notate come e' strutturato il loop nel quale vengono eseguite le blittate.
Gli indirizzi sorgente e destinazione vengono caricati in 2 registri dati
del processore che vengono usati come variabili. Ad ogni iterazione vengono
modificati per puntare al bit-plane seguente. Per questo viene usata la formula

```
INDIRIZZO2 = INDIRIZZO1+2*H*V
```

che avevamo visto a lezione. Nel nostro esempio, V=256 (il numero di righe)
e H=20 (la larghezza dello schermo in words).

In questo esempio la sorgente e la destinazione della blittata sono contenute
nello stesso schermo. Per questo motivo il modulo e' uguale per entrambe,
ed e' calcolato secondo la solita formula.

Lezione9f2

```
; Lezione9f2.s  Scrittura caratteri col blitter

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

;       Puntiamo la PIC "vuota"
        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes (qua sono 2)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
```

```
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        LEA     TESTO(PC),A0    ; Indirizzo del testo da stampare in a0
        LEA     BITPLANE,A3     ; Indirizzo del bitplane destinazione in a3
        bsr.w   Stampa          ; Stampa le linee di testo sullo schermo

        LEA     TESTO2(PC),A0   ; Indirizzo del testo da stampare in a0
        LEA     BITPLANE2,A3    ; Indirizzo del bitplane destinazione in a3
        bsr.w   Stampa          ; Stampa le linee di testo sullo schermo

mouse:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse:

        rts


;*****************************************************************************
;       Routine che stampa caratteri larghi 16x20 pixel
;
;       A0 = punta alla mappa che contiene i caratteri da stampare
;       A3 = punta al bitplane su cui stampare
;*****************************************************************************

STAMPA:
        MOVEQ   #10-1,D3        ; NUMERO RIGHE DA STAMPARE: 10

PRINTRIGA:
        MOVEQ   #20-1,D0        ; NUMERO COLONNE PER RIGA: 20

PRINTCHAR2:
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        ADD.L   D2,D2           ; MOLTIPLICA PER 2 IL NUMERO PRECEDENTE,
                                ; perche' ogni carattere e' largo 16 pixel.
                                ; In questo modo troviamo l'offset.
        MOVE.L  D2,A2

        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        btst    #6,$02(a5)      ; aspetta che il blitter finisca
waitblit:
        btst    #6,$02(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0: copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo
```

```
        move.l  a2,$50(a5)      ; BLTAPT: indirizzo font (sorgente A)
        move.l  a3,$54(a5)      ; BLTDPT; indirizzo bitplane (destinazione D)
        move    #120-2,$64(a5)  ; BLTAMOD: modulo font
        move    #40-2,$66(a5)   ; BLTDMOD: modulo bit planes
        move    #(20<<6)+1,$58(a5) ; BLTSIZE: 16 pixel, ossia 1 word di larg.
                                   ; * 20 linee di altezza. Da notare che per
                                   ; shiftare il 20 si e' usato il comodo
                                   ; simbolo <<, che shifta a sinistra.
                                   ; (20<<6) e' equivalente a (20*64).

        ADDQ.w  #2,A3           ; A3+2,avanziamo di 16 bit (PROSSIMO CARATTERE)

        DBRA    D0,PRINTCHAR2   ; STAMPIAMO D0 (20) CARATTERI PER RIGA

        ADD.W   #40*19,A3       ; ANDIAMO A CAPO
                                ; ci spostiamo in basso di 19 righe.

        DBRA    D3,PRINTRIGA    ; FACCIAMO D3 RIGHE
        RTS



; Attenzione! nel font sono disponibili solo questi caratteri:
;
; !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ

                ; numero caratteri per linea: 20
TESTO:          ;         11111111112
                ;12345678901234567890
        dc.b    ' PRIMA RIGA TESTO 1 ' ; 1
        dc.b    '                    ' ; 2
        dc.b    '     /   /          ' ; 3
        dc.b    '    /   /           ' ; 4
        dc.b    '                    ' ; 5
        dc.b    'S S A R G           ' ; 6
        dc.b    '                    ' ; 7
        dc.b    '                    ' ; 8
        dc.b    'FABIO CIUCCI        ' ; 9
        dc.b    '                    ' ; 10

        EVEN


                ; numero caratteri per linea: 20
TESTO2:         ;         11111111112
                ;12345678901234567890
        dc.b    '                    ' ; 1
        dc.b    'SECONDA RIGA TESTO 2' ; 2
        dc.b    '     /   /          ' ; 3
        dc.b    '    /   /           ' ; 4
        dc.b    '                    ' ; 5
        dc.b    'SESTA RIGA          ' ; 6
        dc.b    '                    ' ; 7
        dc.b    '                    ' ; 8
        dc.b    'F B O C U C         ' ; 9
        dc.b    '    AMIGA RULEZ     ' ; 10

        EVEN

;*****************************************************************************
```

```
        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$2200      ; bplcon0 - 2 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
BPLPOINTERS2:
        dc.w $e4,0,$e6,0        ;secondo bitplane

        dc.w    $180,$000       ; color0 - SFONDO
        dc.w    $182,$19a       ; color1 - SCRITTE primo bitplane
        dc.w    $184,$f62       ; color2 - SCRITTE secondo bitplane
        dc.w    $186,$1e4       ; color3 - SCRITTE primo+secondo bitplane

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;***************************************************************************

; Qui e' memorizzato il FONT di caratteri 16x20. IN CHIP RAM, perche' e'
; copiato col blitter, e non col processore!

FONT:
        incbin  "assembler2:sorgenti6/font16x20.raw"

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres
BITPLANE2:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio usiamo il blitter per stampare caratteri sullo schermo.
Vengono stampate 10 righe di 20 caratteri ciascuna.
Come sorgente abbiamo un font formato da un solo bitplane.
Lo schermo destinazione, invece, e' formato da 2 bitplanes: in questo modo
abbiamo a disposizione 3 colori per i caratteri (cioe' i colori 1,2 e 3,
perche' il colore 0 serve per lo sfondo).
Per stampare un carattere con il colore 1, lo copiamo solo nel bitplane 1, per
stamparlo con il colore 2 lo copiamo solo nel bitplane 2 e per stamparlo con
il colore 3 lo copiamo in entrambi i bitplanes.
Abbiamo fatto una cosa analoga nella Lezione6h.s usando il font 8x8.
La stampa avviene un bitplane per volta. Il testo da stampare e' contenuto
in 2 "mappe" ascii (una per bitplane) alle label TESTO e TESTO2.
Ogni "mappa" o paginata ascii sara' convertita byte per byte nell'offset da
aggiungere all'indirizzo del font per sapere quale carattere stampare.
```

```
Il lavoro e‘ svolto dalla routine Stampa, che viene richiamata una volta per
ogni bitplane.
La routine e‘ composta da 2 cicli annidati (messi uno dentro l’altro).
Il ciclo piu‘ interno stampa una riga di caratteri da sinistra verso destra.
Il ciclo esterno ripete il ciclo interno 10 volte, facendo stampare dunque 10
righe in totale.
Esaminiamo ora in dettaglio come avviene la blittata.
Utiliziamo un font di 60 caratteri 16*20.
Il font e‘ contenuto in un bitplane "invisibile" (perche‘ non lo facciamo
puntare dai BPLxPT) largo 960 pixel e alto 20 righe, nel quale sono disegnati
tutti e 60 i caratteri uno di fianco all’altro (infatti 60*16=960) in
quest’ordine (quello ASCII):

 !"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ

E’ presente il file Font16x20.iff che e’ la figura originale del font.
Attenzione al fatto che mancano i caratteri dopo il sessantesimo:

        [\]^_‘abcdefghijklmnopqrstuvwxyz{|}~

Se volete le lettere minuscole e altri simboli, fatevi un vostro font e una
vostra routine in grado di leggerlo. Fatevi il vostro "standard".
Poiche‘ i font sono larghi 1 word (16 pixel) e alti 20 righe, tale sara‘ la
dimensione della blittata. I moduli sono calcolati con la solita formula.
Il bitplane sorgente e‘ largo 60 words (cioe‘ 960 pixel, cioe‘ 120 bytes)
e quindi il modulo sorgente vale 2*(60-1)=120-2=118.
Il bitplane destinazione e‘ largo 20 words (cioe‘ 320 pixel, cioe‘ 40 bytes)
e quindi il modulo sorgente vale 2*(20-1)=40-2=38.
Vediamo come vengono gestiti i puntatori. Il puntatore alla destinazione varia
ad ogni blittata per disegnare il carattere in una diversa posizione dello
schermo, procedendo da sinistra a destra e dall’alto in basso.
Il meccanismo e‘ lo stesso che abbiamo visto nell’esempio lezione9c2.s.
Il puntatore alla sorgente invece deve puntare ogni volta al carattere che deve
essere stampato. I dati del bitplane sorgente sono organizzati cosi‘:

INDIRIZZO       CONTENUTO
FONT            prima riga(16 pixel quindi 1 word) del carattere ’ ’
FONT+2          prima riga del carattere ’!’
FONT+4          prima riga del carattere ’"’

.
.
.
FONT+120        prima riga del carattere ’Z’
FONT+122        seconda riga del carattere ’ ’
FONT+124        seconda riga del carattere ’!’
.
.
.

FONT+2282       ultima riga del carattere ’ ’
FONT+2284       ultima riga del carattere ’!’
.
FONT+2398       ultima riga del carattere ’Z’


La routine legge dalla mappa il codice ASCII del carattere che deve stampare
e da esso calcola l’indirizzo. Il metodo e‘ molto simile a quello che abbiamo
visto nella lezione 6 quando abbiamo fatto la stessa cosa con il processore.
Dal codice ASCII possiamo ricavare la distanza del carattere dall’inizio del
font. Per farlo prima sottraiamo 32 (cioe‘ il codice ASCII dello spazio) al
codice ASCII del carattere che stamperemo, perche‘ il primo carattere del font
```

e‘ lo spazio. A questo punto procediamo diversamente dal metodo della lezione
6. Infatti il font della lezione 6 era disegnato "in verticale", cioe‘:

```
!
"
#

ecc.

>
?
@
A
B
C
D
E
F
G

ecc.
```

In quel caso, quindi per calcolare l’indirizzo dovevamo moltiplicare il codice ASCII (meno 32) per la quantita‘ di memoria occupata da un carattere.
In questo caso, invece, il font e‘ disegnato "in orizzontale", siccome a noi interessa l’indirizzo della prima word del carattere da disegnare, dovremo moltiplicare il codice ASCII (meno 32) per la quantita‘ di memoria occupata dalla PRIMA RIGA di ogni carattere, in quanto la prima riga del carattere che ci interessa e‘ memorizzata DOPO la prima riga dei caratteri che lo precedono ma PRIMA di tutte le altre righe (a differenza del caso della lezione 6, nel quale tutte le righe di un carattere erano memorizzate prima del carattere successivo). Siccome una riga occupa 2 byte (1 word = 16 pixel) dobbiamo moltiplicare per 2, cosa che possiamo fare con una semplice ADD, risparmiandoci una lenta MULU.

Lezione9f3

```
; Lezione9f3.s          In questo listato una figura di 16*15 pixel, a 2
;                       bitplanes, viene blittata ripetutamente fino a
;                       riempire lo schermo (320*256 lowres 2 bitplanes).
;                       La temporizzazione col Wblank fa in modo che venga
;                       blittata solo una mattonella per fotogramma.

        section bau,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo il primo bitplane
```

Figura 25.5: Lezione 9f2

```
        MOVE.L  #BitPlane1,d0   ; dove puntare
        LEA     BPLPOINTER1,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo il secondo bitplane

        MOVE.L  #BitPlane2,d0   ; dove puntare
        LEA     BPLPOINTER2,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.s   fillmem                 ; riempi lo schermo di "mattonelle"
                                        ; col blitter.
mouse:
        btst    #6,$bfe001              ; testa il tasto sin. del mouse
        bne.s   mouse
        rts                             ; uscita

fillmem:
        lea     Bitplane1,a0    ; primo bitplane
        lea     Bitplane2,a1    ; secondo bitplane
        lea     gfxdata1,a3     ; fig. plane 1
```

```
        lea     gfxdata2,a4     ; fig. plane 2

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo
        move.w  #0,$64(a5)              ; BLTAMOD = 0
        move.w  #38,$66(a5)             ; BLTDMOD (40-2=38), infatti ogni
                                        ; "mattonella" e' larga 16 pixel,
                                        ; cioe' 2 bytes, che dobbiamo togliere
                                        ; alla larghezza totale di una linea,
                                        ; cioe' 40, e il risultato e' 40-2=38!
        move.w  #$0000,$42(a5)          ; BLTCON1 - lo spiegheremo dopo
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)

        moveq   #16-1,d7                ; 16 file di blocchi per arrivare
                                        ; verticalmente fino in fondo, infatti
                                        ; le mattonelle sono alte 15 pixel,
                                        ; piu' 1 di "spaziatura" tra una e
                                        ; l'altra, sotto ognuna, fa un
                                        ; ingombro di 16 pixel per piastrella,
                                        ; dunque 256/16=16 mattonelle.
FaiTutteLeRighe:
        moveq   #20-1,d6                ; 20 blocchi per linea (fila), infatti,
                                        ; essendo le mattonelle larghe 16
                                        ; pixel, cioe' 2 bytes, ne deriva che
                                        ; ce ne possono stare 320/16=20 per
                                        ; linea orizzontale.
FaiUnaRigaLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

; Blitta il primo bitplane di una mattonella

        move.l  a0,$54(a5)              ; BLTDPT - destinazione (bitpl 1)
        move.l  a3,$50(a5)              ; BLTAPT - sorgente (fig1)
        move.w  #(15*64)+1,$58(a5)      ; BLTSIZE - altezza 15 words,
                                        ;           larghezza 1 word
                                        ; Per Fare il Primo Bitplane

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

; Blitta il secondo bitplane di una mattonella

        move.l  a1,$54(a5)              ; BLTDPT - destinazione (bitpl 2)
        move.l  a4,$50(a5)              ; BLTAPT - sorgente (fig2)
        move.w  #(15*64)+1,$58(a5)      ; BLTSIZE - altezza 15 words,
```

```
                                    ;           larghezza 1 word
                                    ; Per Fare il Primo Bitplane

        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        addq.w  #2,a0   ; salta 1 word (16 pixel) nel bitplane 1, scattando
                        ; in "avanti" per la prossima mattonella
        addq.w  #2,a1   ; salta 1 word (16 pixel) nel bitplane 2
        dbra    d6,FaiUnaRigaLoop       ; e cicla fino a che non sono state
                                        ; blittate tutte le 20 mattonelle
                                        ; di una riga.

        lea     15*40(a0),a0    ; salta 15 linee nel bitplane 1. Siccome
                                ; abbiamo gia' incrementato a0 a forza di
                                ; addq #2,a0, abbiamo gia' saltato una linea
                                ; intera prima di arrivare qua. Per ogni loop,
                                ; dunque, vengono saltate 16 linee, lasciando
                                ; tra una mattonella e l'altra una "striscia"
                                ; di sfondo azzerato, dato che le mattonelle
                                ; sono alte solo 15 pixel.
        lea     15*40(a1),a1    ; salta 15 linee nel bitplane 2
        dbra    d7,FaiTutteLeRighe      ; fai tutte le 16 righe

        rts

;******************************************************************************

                section cop,data_C

copperlist
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w $100,$2200         ; BPLCON0 - 2 bitplanes lowres

        dc.w $180,$000  ; Color0
        dc.w $182,$FED  ; Color1
        dc.w $184,$33a  ; Color2
        dc.w $186,$888  ; Color3

BPLPOINTER1:
        dc.w $e0,0,$e2,0        ;primo   bitplane
BPLPOINTER2:
        dc.w $e4,0,$e6,0        ;secondo bitplane

        dc.l    $ffff,$fffe     ; fine della copperlist

*******************************************************************************

;       Figura, composta da 2 biplanes. larghezza = 1 word, altezza = 15 linee

gfxdata1:
        dc.w    %1111111111111100
```

```
        dc.w    %1111111111111100
        dc.w    %1100000000001100
        dc.w    %1101111111111100
        dc.w    %1101111111111100
        dc.w    %1101111111011100
        dc.w    %1101110011011100
        dc.w    %1101110111011100
        dc.w    %1101111111011100
        dc.w    %1101111111011100
        dc.w    %1101100000011100
        dc.w    %1101111111111100
        dc.w    %1111111111111100
        dc.w    %1111111111111100
        dc.w    %0000000000000000

gfxdata2:
        dc.w    %0000000000000010
        dc.w    %0111111111111110
        dc.w    %0111111111110110
        dc.w    %0111111111110110
        dc.w    %0111000000010110
        dc.w    %0111011111110110
        dc.w    %0111011101110110
        dc.w    %0111011101110110
        dc.w    %0111010001110110
        dc.w    %0111011111110110
        dc.w    %0111011111110110
        dc.w    %0111111111110110
        dc.w    %0100000000000110
        dc.w    %0111111111111110
        dc.w    %1111111111111110

;******************************************************************************

        section gnippi,bss_C

bitplane1:
                ds.b    40*256
bitplane2:
                ds.b    40*256

        end

;******************************************************************************

Questo esempio e‘ una variazione dell’esempio lezione9c2.s.
Questa volta abbiamo uno schermo da 2 planes.
Anche le nostre mattonelle sono costituite da 2 planes.
La routine che esegue la "piastrellatura" dello schermo, ha la stessa
struttura di quella dell’esempio lezione9c2.s, solo che vengono effettuate 2
copie: il primo bitplane della mattonella sul primo bitplane dello schermo e
il secondo bitplane della mattonella sul secondo dello schermo.
Inoltre, tanto per renderlo piu‘ interessante, abbiamo rallentato la routine
mettendo un loop di attesa del Vertical Blank.
In questo modo, le mattonelle vengono copiate una ogni Vertical Blank, ed e‘
possibile osservare ad occhio l’ordine in cui vengono copiate.
```

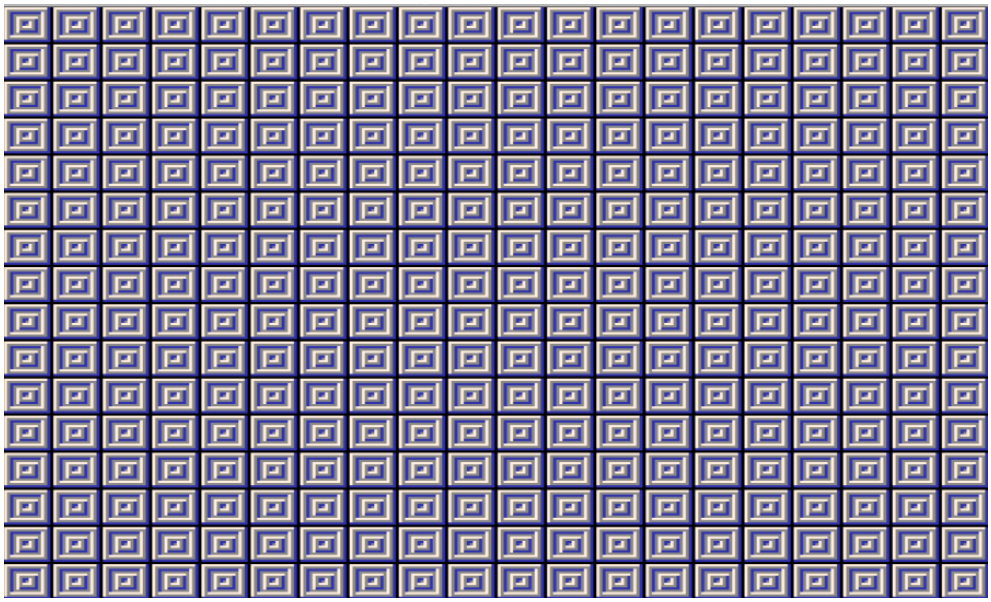


Figura 25.6: Lezione 9f3

25.7 Lezione9g1

```
; Lezione9g1.s  Visualizzazione di un'immagine INTERLEAVED
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA PRIMA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP
```

```
        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse:
        rts

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' LA SECONDA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                ; qua carichiamo la figura in
                                ; formato RAWBLIT (o interleaved),
                                ; convertita col KEFCON.
        end
```

In questo esempio visualizziamo un'immagine in formato interleaved (o rawblit, come la chiama il KEFCON). Si tratta della solita immagine, ma abbiamo dovuto convertirla nel formato interleaved, pertanto usiamo un file diverso.
Come abbiamo gia' detto nella lezione, per visualizzare immagini in questo formato bisogna cambiare 2 cose rispetto ad immagini normali:
1) Nel puntare i bitplane, bisonga calcolare diversamente gli indirizzi dei

```
vari bitplane che "distano" tra loro di una sola riga, e non di tutte le righe
del bitplane;
2) i moduli dei bitplane non valgono 0, ma servono per "saltare" le righe degli
altri bitplanes. Sono calcolati mediante la formula che abbiamo visto:

 MODULO=2*L*(N-1)       Dove L e' la larghezza del bitplane espressa in words
                        e N e' il numero di bitplanes

Nel nostro caso i bitplane sono larghi 20 words (320/16) ovvero 40 bytes,
e il numero di bitplanes e' 3. Potete trovare la differenza 1) nelle prime
linee del listato, nel loop che punta i bitplane nella copperlist, e la
differenza 2) nelle istruzioni della copperlist che caricano il valore del
modulo in BPL1MOD e BPL2MOD.
```

Lezione9g2

```
; Lezione9g2.s  BLITTATA, in cui copiamo un rettangolo da un punto
;               all'altro dello stesso schermo in formato INTERLEAVED
;               Tasto sinistro per eseguire la blittata, destro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta
```

```
        bsr.s   copia           ; esegui la routine di copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


; ********************** LA ROUTINE DI COPIA ***************************

 ; viene copiato un rettangolo con larghezza=160 e altezza=20
 ; dalle coordinate X1=64, Y1=50 (sorgente)
 ; alle coordinate X2=80, Y2=190 (destinazione)

copia:

; Carica gli indirizzi sorgente e destinazione in 2 registri
; NOTATE LA DIFFERENZA RISPETTO AL CASO NORMALE: NEL CALCOLO DELL'OFFSET
; DELLA RIGA Y SI MOLTIPLICA ANCHE PER IL NUMERO DI PLANES (cioe' 3)
; la formula e' OFFSET=(Y*(NUMERO WORDS PER RIGA)*(NUMERO PLANES))*2

        move.l  #bitplane+((20*3*50)+64/16)*2,d0        ; ind. sorgente
                                                        ; notate il fattore *3!
        move.l  #bitplane+((20*3*190)+80/16)*2,d2       ; ind. destinazione
                                                        ; notate il fattore *3!

        btst    #6,2(a5)        ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo

; carica i puntatori
        move.l  d0,$50(a5)      ; bltapt
        move.l  d2,$54(a5)      ; bltdpt

; queste 2 istruzioni settano i moduli della sorgente e della destinazione
; NON CI SONO DIFFERENZE RISPETTO AL CASO NORMALE:
; il modulo e' calcolato secondo la formula (H-L)*2  (H e' la larghezza del
; bitplane in words e L e' la larghezza dell'immagine, sempre in words)
; che abbiamo visto a lezione, (20-160/16)*2=20

        move.w  #(20-160/16)*2,$64(a5)  ; bltamod
        move.w  #(20-160/16)*2,$66(a5)  ; bltdmod

; notate anche che poiche' i 2 registri hanno indirizzi consecutivi, si puo'
; usare una sola istruzione invece che 2 (ricordate che 20=$14):
; move.l #$00140014,$64(a5)     ; bltamod e bltdmod

; NOTATE LA DIFFERENZA RISPETTO AL CASO NORMALE: NELLA DIMENSIONE
; DELLA BLITTATA, L'ALTEZZA DELL'IMMAGINE E' MOLTIPLICATA PER IL NUMERO
; DI BITPLANES

        move.w  #(3*20*64)+160/16,$58(a5)       ; bltsize
                                                ; altezza 20 linee e 3 planes
                                                ; largo 160 pixel (= 10 words)

        btst    #6,$02(a5)      ; aspetta che il blitter finisca
waitblit2:
```

```
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                ; qua carichiamo la figura in
                                ; formato RAWBLIT (o interleaved),
                                ; convertita col KEFCON.
        end

;***************************************************************************

In questo esempio visualizziamo un'immagine in formato interleaved e ne copiamo
un pezzo da un punto all'altro dello schermo. Si tratta dello stesso programma
dell'esempio lezione9f1.s, ma in formato interleaved.
Vi consigliamo di esaminare questo esempio confrontandolo con lezione9f1.s.
Come abbiamo visto nella lezione, il formato interleaved ci permette di
effettuare la copia mediante una sola blittata. Per questo la routine "Copia"
(che e' la routine che effettua la copia) e' priva di loop, a differenza
dell'omonima routine di lezione9f1.s.
Alcuni valori caricati nei registi del blitter sono diversi:

1) Nel calcolo dell'indirizzo, per ottenere l'offset tra la prima word della
   riga Y e l'inizio del bitplane, bisogna moltiplicare Y per il numero dei
```

```
   bitplanes, oltre che per la dimensione della riga; per quanto riguarda la
   X, invece, non ci sono differenze.

2) L'altezza della blittata e' pari all'altezza dell'immagine moltiplicata per
   il numero dei bitplanes; per quanto riguarda la larghezza, invece, non ci
   sono differenze.

Anche per quanto riguarda gli altri registri, in particolare per il modulo,
non ci sono differenze.
```

Lezione9g3

```
; Lezione9g3.s  Ancora le mattonelle, stavolta pero' con lo schermo INTERLEAVED
;               La temporizzazione col Wblank fa in modo che venga
;               blittata solo una FILA per fotogramma.
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes (qua sono 2)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.s   fillmem         ; esegui la routine di "piastrellatura"

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts
```

```
*****************************************************************************
; Routine che esegue la piastrellatura
*****************************************************************************

fillmem:
        lea     Bitplane,a0     ; bitplanes
        lea     gfxdata,a3      ; ind. figura

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo
        move.w  #0,$64(a5)              ; BLTAMOD = 0
        move.w  #38,$66(a5)             ; BLTDMOD (40-2=38), infatti ogni
                                        ; "mattonella" e' larga 16 pixel,
                                        ; cioe' 2 bytes, che dobbiamo togliere
                                        ; alla larghezza totale di una linea,
                                        ; cioe' 40, e il risultato e' 40-2=38!
        move.w  #$0000,$42(a5)          ; BLTCON1 - lo spiegheremo dopo
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)

        moveq   #16-1,d7                ; 16 file di blocchi per arrivare
                                        ; verticalmente fino in fondo, infatti
                                        ; le mattonelle sono alte 15 pixel,
                                        ; piu' 1 di "spaziatura" tra una e
                                        ; l'altra, sotto ognuna, fa un
                                        ; ingombro di 16 pixel per piastrella,
                                        ; dunque 256/16=16 mattonelle.
FaiTutteLeRighe:
        moveq   #20-1,d6                ; 20 blocchi per linea (fila), infatti,
                                        ; essendo le mattonelle larghe 16
                                        ; pixel, cioe' 2 bytes, ne deriva che
                                        ; ce ne possono stare 320/16=20 per
                                        ; linea orizzontale.

; aspetta il vblank una volta ogni riga disegnata.
WaitWblank:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

FaiUnaRigaLoop:

; Blitta il primo bitplane di una mattonella

        move.l  a0,$54(a5)              ; BLTDPT - destinazione (bitpl 1)
        move.l  a3,$50(a5)              ; BLTAPT - sorgente (fig1)
        move.w  #(2*15*64)+1,$58(a5)    ; BLTSIZE - altezza: 2 planes
                                        ; alti 15 righe
                                        ; larghezza 1 word
```

```
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2


        addq.w  #2,a0   ; salta 1 word (16 pixel) nel bitplane 1, scattando
                        ; in "avanti" per la prossima mattonella
        dbra    d6,FaiUnaRigaLoop       ; e cicla fino a che non sono state
                                        ; blittate tutte le 20 mattonelle
                                        ; di una riga.

        lea     40+2*15*40(a0),a0
                                ; A forza di ADDQ #2,A0, abbiamo incrementato
                                ; il puntatore a0 fino a superare l'ultima word
                                ; della riga 0, plane 1. Quindi siamo arrivati
                                ; alla prima word della riga 0, plane 2.
                                ; Ora ci vogliamo spostare alla prima word
                                ; della riga 16, plane 1. Dobbiamo quindi
                                ; sommare ad A0: 40 per spostarci sulla prima
                                ; word della riga 1, plane 1 e poi 2*15*40
                                ; per spostarci dove vogliamo.

        dbra    d7,FaiTutteLeRighe      ; fai tutte le 16 righe
        rts

*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,40         ; VALORE MODULO = 2*20*(2-1)= 40
        dc.w    $10a,40         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$2200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000

        dc.w $180,$000  ; Color0
        dc.w $182,$FED  ; Color1
        dc.w $184,$33a  ; Color2
        dc.w $186,$888  ; Color3

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;       Figura, composta da 2 biplanes. larghezza = 1 word, altezza = 15 linee

*****************************************************************************
; Figura della mattonella

; Si tratta della stessa figura dell'esempio lezioine9f3.s solo che li'
```

```
; era in formato normale. Per metterla in formato interleaved sono state
; "mischiate" le righe.

gfxdata:
        dc.w    %1111111111111100       ; riga 0, plane 1
        dc.w    %0000000000000010       ; riga 0, plane 2
        dc.w    %1111111111111100       ; riga 1, plane 1
        dc.w    %0111111111111110       ; riga 1, plane 2
        dc.w    %1100000000001100       ; riga 3, plane 1
        dc.w    %0111111111110110       ; riga 3, plane 2
        dc.w    %1101111111111100
        dc.w    %0111111111110110
        dc.w    %1101111111111100
        dc.w    %0111000000010110
        dc.w    %1101111111011100
        dc.w    %0111011111110110
        dc.w    %1101110011011100
        dc.w    %0111011101110110
        dc.w    %1101110111011100
        dc.w    %0111011101110110
        dc.w    %1101111111011100
        dc.w    %0111010001110110
        dc.w    %1101111111011100
        dc.w    %0111011111110110
        dc.w    %1101100000011100
        dc.w    %0111011111110110
        dc.w    %1101111111111100
        dc.w    %0111111111110110
        dc.w    %1111111111111100
        dc.w    %0100000000000110
        dc.w    %1111111111111100
        dc.w    %0111111111111110
        dc.w    %0000000000000000       ; riga 15, plane 1
        dc.w    %1111111111111110       ; riga 15, plane 2

*****************************************************************************

        section gnippi,bss_C

bitplane:
                ds.b    2*40*256        ; 2 bitplanes

        end

*****************************************************************************

In questo esempio ritroviamo le mattonelle, stavolta in formato interleaved.
Come prima cosa notate come viene disposta in memoria la figura della
mattonella. Nell'esempio lezione9f3.s avevamo i 2 bitplanes separati.
Qui invece le righe sono mescolate tra loro.
Per quanto riguarda la routine, le mattonelle vengono copiate con una sola
blittata, mentre in lezione9f3.s dovevamo fare una blittata per ogni bitplane.
L'altezza della blittata e' pari al prodotto dell'altezza della figura
(15 linee) per il numero di bitplanes (2), come abbiamo spiegato nella lezione.
Anche il calcolo dell' indirizzo destinazione e' (naturalmente) diverso
(la sorgente e' fissa e pertanto resta sempre uguale). Le mattonelle sulla
stessa riga si trovano distanziate sempre di una word, come e' logico visto
che l'interleaved si differenzia solo per la disposizione delle righe.
La differenza si trova dopo il loop interno, quando siamo arrivati alla fine
di una fila orizzontale di mattonelle e dobbiamo iniziare la seguente.
Se indichiamo con Y la riga alla quale iniziamo a blittare, dobbiamo spostarci
alla riga Y+16.
```

```
Nel loop interno, aumentiamo il puntatore di 2 ogni volta, spostandoci in tal
modo di una word verso destra. Alla fine del loop ci troviamo subito dopo
l'ultima word della riga attuale, cioe' sulla prima word del plane 2
della riga Y. Per prima cosa dobbiamo spostarci sul plane 1 della riga Y+1,
aggiungendo 40 (numero di bytes occupati da UN plane di UNA riga).
A questo punto dobbiamo scendere di altre 15 righe.
Poiche' un plane di una riga occupa 40 bytes, e noi abbiamo 2 planes per ogni
riga, dobbiamo aggiungere 2*15*40.
Naturalmente, possiamo aggiungere in una volta sola entrambe le quantita'
facendo una LEA 40+15*2*40(A1),A1.

Riassumiamo la situazione con la seguente figura:

- All'inizio del loop interno il puntatore punta la word indicata con (0).
- Alla fine del loop interno il puntatore punta la word indicata con (1).
- Aggiungendo 40 il puntatore punta la word indicata con (2).
- Aggiungendo 2*40*15 ci spostiamo in basso di 15 righe e il puntatore punta
  la word indicata con (3) che e' la word che volevamo.
  (tra una riga e l'altra ci sono 2*40 bytes; se avessimo aggiunto solo 2*40
   ci saremmo spostati dalla word (2) alla word (2')).

riga Y     plane 1      | (0)  |      |      |    . . .   |      |
riga Y     plane 2      | (1)  |      |      |    . . .   |      |
riga Y+1   plane 1      | (2)  |      |      |    . . .   |      | \
riga Y+1   plane 2      |      |      |      |    . . .   |      |  |
riga Y+1   plane 1      | (2') |      |      |    . . .   |      |  |
riga Y+1   plane 2      |      |      |      |    . . .   |      |  |
                                                                  |
.                                                                 |
                                                                  | 15 righe
.                                                                 |
                                                                  |
.                                                                 |
                                                                  |
                                                                 /
riga Y+16  plane 1      | (3)  |      |      |    . . .   |      |
riga Y+16  plane 2      |      |      |      |    . . .   |      |
```

25.8 Lezione9h1

```
; Lezione9h1.s          Esempio di uso delle Mask

; Premete alternativamente i tasti destro e sinistro del mouse per vedere
; varie blittate con maschere diverse.


        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"
```

```
        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; prepara i parametri

        move.w  #$ffff,d0               ; maschera prima word
                                        ; fa passare tutti i bit
        move.w  #$ffff,d1               ; maschera ultima word
                                        ; fa passare tutti i bit
        lea     bitplane+2,a0           ; indirizzo destinazione
        bsr.w   Copia

mouse2:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse2

; prepara i parametri

        moveq   #$0000,d0               ; maschera prima word
                                        ; cancella tutto
        move.w  #$ffff,d1               ; maschera ultima word
                                        ; fa passare tutti i bit
        move.l  #bitplane+30*40+2,a0    ; indirizzo destinazione
        bsr.s   Copia
mouse3:
        btst    #6,$bfe001              ; mouse premuto?
        bne.s   mouse3

; prepara i parametri

        move.w  #%1010101010101010,d0   ; maschera prima word
                                        ; un bit si e uno no
        move.w  #%0000000000000001,d1   ; maschera ultima word
                                        ; solo bit piu' a destra
        lea     bitplane+60*40+2,a0     ; indirizzo destinazione
        bsr.s   Copia

mouse4:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse4

; prepara i parametri

        moveq   #$0000,d0               ; maschera prima word
                                        ; cancella tutto
        moveq   #$0000,d1               ; maschera ultima word
                                        ; cancella tutto
        lea     bitplane+90*40+2,a0     ; indirizzo destinazione
        bsr.s   Copia
```

```
mouse5:
        btst    #6,$bfe001              ; mouse premuto?
        bne.s   mouse5

; prepara i parametri
        move.w  #%1111000011110000,d0   ; maschera prima word
                                        ; 4 bit si e 4 no
        move.w  #%0000011010011100,d1   ; maschera ultima word
                                        ; fa passare solo i bit 2,3,4,7,9 e 10
        lea     bitplane+120*40+2,a0    ; indirizzo destinazione
        bsr.s   Copia

mouse6:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse6

; prepara i parametri
        move.w  #%0000000001111111,d0   ; maschera prima word
                                        ; cancella i 9 bit piu' a sinistra
        move.w  #%1111111000000000,d1   ; maschera ultima word
                                        ; cancella i 9 bit piu' a destra
        lea     bitplane+150*40+2,a0    ; indirizzo destinazione
        bsr.s   Copia

mouse:
        btst    #6,$bfe001
        bne.s   mouse

        rts

;*****************************************************************************
; Questa routine copia la figura sullo schermo.
;
; A0   - indirizzo destinazione
; D0.w - maschera prima word
; D1.w - maschera ultima word
;*****************************************************************************

Copia:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  d0,$44(a5)              ; BLTAFWM carica il parametro
        move.w  d1,$46(a5)              ; BLTALWM carica il parametro
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 lo spieghiamo dopo
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*7)+3,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
        rts                             ; altezza 7 linee (1 plane)

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
```

```
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $100,$1200      ; bplcon0 - 1 bitplane Lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Definiamo in binario la figura, che e' larga 3 words, e alta 7 linee

Figura:
;                0123456789012345  0123456789012345  0123456789012345
        dc.w    %1111111111000000,%0000001111000000,%0000001111111111
        dc.w    %1111111111000000,%0000111111110000,%0000001111111111
        dc.w    %1111111111000000,%0011111111111100,%0000001111111111
        dc.w    %1111111111111111,%1111111111111111,%1111111111111111
        dc.w    %1111111111000000,%0011111111111100,%0000001111111111
        dc.w    %1111111111000000,%0000111111110000,%0000001111111111
        dc.w    %1111111111000000,%0000001111000000,%0000001111111111

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio mostriamo come le maschere influiscano sulla blittata.
Abbiamo una figura (ad un solo plane) che viene copiata piu‘ volte sullo
schermo in posizioni diverse.
Ogni copia, pero‘ viene eseguita con delle maschere diverse, producendo gli
effetti che vedete.
Abbiamo utilizzato una routine che prende come parametri i valori delle
maschere e l’indirizzo destinazioine della blittata.
In tal modo eseguiamo tutte le copie con una sola routine, cambiando solo il
valore dei parametri.
I parametri vengono passati alla routine attraverso dei registri del 68000.
Per capire l’effetto che hanno i diversi valori delle maschere, leggete i
commenti nel listato.
```

Lezione9h2

```
; Lezione9h2.s  BLITTATA, in cui copiamo un rettangolo (in una pic normale)
;               non allineato con una word, usando le maschere per "tappare".
```

Figura 25.7: Lezione 9h1

```
;               Premendo il tasto destro si esegue la blittata "sporca"
;               Poi,premendo il tasto sinistro si esegue la blittata giusta
;               e infine premendo ancora il tasto destro si usce.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
```

```
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse1                  ; se no, aspetta

; Prima Blittata, con le maschere che lasciano passare anche dati non
; desiderati

        lea     bitplane+((20*170)+80/16)*2,a0          ; ind. destinazione
        move.w  #$ffff,d0                               ; passa tutto
        move.w  #$ffff,d1                               ; passa tutto
        bsr.s   copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

; Seconda blittata, grazie alle maschere viene copiata solo
; lettera "I"

        lea     bitplane+((20*170)+160/16)*2,a0         ; ind. destinazione
        move.w  #%0000000000001111,d0   ; passano i 4 bit piu' a destra
        move.w  #%1111000000000000,d1   ; passano i 4 bit piu' a sinistra
        bsr.s   copia

mouse3:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse3          ; se no, aspetta

        rts

;*****************************************************************************
; Questa routine copia la figura sullo schermo.
;
; A0   - indirizzo destinazione
; D0.w - maschera prima word
; D1.w - maschera ultima word
;*****************************************************************************

;         ___________
;        (_____ _____)
;        /(_o(___)O_)\
;       / ___________ \
;       \ \____l____/ /|
;       |\_'---'---'_/ |
;       | '---------'  |
;       |  T  xCz   T  |
;       l__|        l__|
;       (__)---^----(__)
;         T    T     |
;        _l____l_____|_
;       (______X_______)

copia:

        lea     bitplane+((20*78)+128/16)*2,a1  ; indirizzo sorgente fisso

        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
```

```
        btst    #6,2(a5)        ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D

                                        ; carica i parametri nelle maschere
        move.w  d0,$44(a5)              ; BLTAFWM mask a sinistra
        move.w  d1,$46(a5)              ; BLTALWM mask a destra

; carica i puntatori

        move.l  a1,$50(a5)              ; bltapt - sorgente
        move.l  a0,$54(a5)              ; bltdpt - destinazione

        move.l #$00240024,$64(a5)       ; bltamod e bltdmod

        move.w  #(60*64)+2,$58(a5)      ; bltsize
                                        ; altezza 60 linee
                                        ; larghezza 2 words

        lea     40*256(a1),a1           ; punta al prossimo plane sorgente
        lea     40*256(a0),a0           ; punta al prossimo plane destinazione

        dbra    d7,PlaneLoop

        btst    #6,$02(a5)      ; aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; IMMAGINI NORMALI!!!!!!
        dc.w    $108,0          ; VALORE MODULO = 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
```

```
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.raw"
                                        ; qua carichiamo la figura in
                                        ; formato RAW convertita col KEFCON.
        end

;*****************************************************************************

In questo esempio mettiamo in evidenza come mediante l'uso delle maschere
sia possibile estrarre "pezzi" di immagine cancellando delle parti non
desiderate. In questo caso vogliamo copiare solo la lettera "I" della
scritta Amiga. Tale lettera e' contenuta in un rettangolo largo 2 words.
Nel rettangolo pero' ci sono anche pezzi di altre lettere.
La prima blittata viene eseguita con le maschere al valore $ffff, cioe'
settate in modo tale da far passare tutti i pixel.
Come vedete vengono copiati anche i pezzi di altre lettere.
La seconda blittata, invece ha le maschere settate a dei valori opportuni
tali da far passare solo i pixel che formano la lettera "I"
```

Lezione9h2r

```
; Lezione9h2r.s BLITTATA, in cui copiamo un rettangolo (in una pic RAWBLIT)
;               non allineato con una word, usando le maschere per "tappare".
;               Premendo il tasto destro si esegue la blittata "sporca"
;               Poi,premendo il tasto sinistro si esegue la blittata giusta
;               e infine premendo ancora il tasto destro si usce.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
```

```
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse1                  ; se no, aspetta

; Prima Blittata, con le maschere che lasciano passare anche dati non
; desiderati

        lea     bitplane+((20*3*170)+80/16)*2,a0        ; ind. destinazione
        move.w  #$ffff,d0                               ; passa tutto
        move.w  #$ffff,d1                               ; passa tutto
        bsr.s   copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

; Seconda blittata, grazie alle maschere viene copiata solo
; lettera "I"

        lea     bitplane+((20*3*170)+160/16)*2,a0       ; ind. destinazione
        move.w  #%0000000000001111,d0   ; passano i 4 bit piu' a destra
        move.w  #%1111000000000000,d1   ; passano i 4 bit piu' a sinistra
        bsr.s   copia

mouse3:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse3          ; se no, aspetta

        rts

;*****************************************************************************
; Questa routine copia la figura sullo schermo.
;
; A0   - indirizzo destinazione
; D0.w - maschera prima word
; D1.w - maschera ultima word
;*****************************************************************************

;         ___________
;        (_____ _____)
;        /(_o(___)O_)\
;       / ___________ \
;       \ \____l____/ /|
;       |\_'---'---'_/ |
;       | '---------'  |
;       |  T  xCz   T  |
;       l__|        l__|
;       (__)---^----(__)
;         T    T     |
;        _l____l_____|_
;       (______X_______)
```

```
copia:
        btst    #6,2(a5)        ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D

                                        ; carica i parametri nelle maschere
        move.w  d0,$44(a5)              ; BLTAFWM mask a sinistra
        move.w  d1,$46(a5)              ; BLTALWM mask a destra

; carica i puntatori

        move.l  #bitplane+((20*3*78)+128/16)*2,$50(a5) ; bltapt: sorgente fissa
        move.l  a0,$54(a5)                             ; bltdpt: destinazione

        move.l #$00240024,$64(a5)               ; bltamod e bltdmod

        move.w  #(3*60*64)+2,$58(a5)            ; bltsize
                                                ; altezza 60 linee e 3 planes
                                                ; larghezza 2 words

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7
```

```
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.
        end

;***************************************************************************

Questo esempio e' la versione rawblit di lezione9h2.s.
Confrontate le differenze nelle formule per il calcolo dei valori da scrivere
nei registri del blitter.
```

Lezione9h3

```
; Lezione9h3.s  Facciamo apparire un immagine una colonna di pixel alla volta
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta
```

```
        bsr.s   Mostra          ; esegui la routine

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


; ********************** LA ROUTINE CHE MOSTRA LA FIGURA ******************

Mostra:

; valori iniziali dei puntatori

        lea     picture,a0              ; punta all'inizio della figura
        lea     bitplane,a1             ; punta all'inizio del primo bitplane

        moveq   #20-1,d7                ; esegui per ogni "colonna" di word.
                                        ; lo schermo e' largo 20 word, quindi
                                        ; ci sono 20 colonne.

FaiTutteLeWord:
        moveq   #16-1,d6                ; 16 pixel per ogni word.
        move.w  #%1000000000000000,d0   ; valore della maschera all'inizio del
                                        ; loop interno. Fa passare solo il
                                        ; pixel piu' a sinistra della word.
FaiUnaWord:

; aspetta il vblank in modo da disegnare una colonna di pixel ad ogni
; fotogramma.

WaitWblank:
        CMP.b   #$ff,$dff006            ; vhposr - aspetta la linea 255
        bne.s   WaitWblank
Aspetta:
        CMP.b   #$ff,$dff006            ; vhposr - ancora linea 255 ?
        beq.s   Aspetta

        moveq   #3-1,d5                 ; ripeti per ogni plane

        move.l  a0,a2                   ; copia i puntatori in altri 2 registri
        move.l  a1,a3                   ; questo perche' all'interno del loop
                                        ; che disegna i vari plane devono
                                        ; modificare per puntare da un plane
                                        ; all'altro.

FaiUnPlane:
        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
        move.w  #$ffff,$44(a5)          ; BLTAFWM - fa passare tutti i bit
        move.w  d0,$46(a5)              ; Carica il valore della maschera nel
                                        ; registro BLTALWM
; carica i puntatori

        move.l  a2,$50(a5)              ; bltapt
        move.l  a3,$54(a5)              ; bltdpt
```

```
; Sia per la sorgente che per la destinazione blittiamo una word appartenente
; ad uno schermo largo 20 word. Il modulo quindi vale 2*(20-1)=38=$26.
; Poiche' i 2 registri hanno indirizzi consecutivi, si puo' usare una sola
; istruzione invece che 2:

        move.l #$00260026,$64(a5)       ; bltamod e bltdmod

; blittiamo una "colonna" di word alta 256 righe (tutto lo schermo)

        move.w  #(256*64)+1,$58(a5)     ; bltsize
                                        ; altezza 256 linee
                                        ; larghezza 1 word

        lea     40*256(a2),a2           ; punta al prossimo plane sorgente
        lea     40*256(a3),a3           ; punta al prossimo plane destinazione

        dbra    d5,FaiUnPlane           ; ripeti per tutti i planes

        asr.w   #1,d0                   ; calcola la maschera per la prossima
                                        ; blittata. Fa passare ogni volta un
                                        ; bit in piu' rispetto alla volta
                                        ; precedente.

        dbra    d6,FaiUnaWord           ; ripeti per tutti i pixel

        addq.w  #2,a0                   ; punta alla word seguente
        addq.w  #2,a1                   ; punta alla word seguente

        dbra    d7,FaiTutteLeWord       ; ripeti per tutte le word

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2

        rts

;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO = 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
```

```
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

PICTURE:
        incbin  "assembler2:sorgenti6/amiga.raw"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.

;***************************************************************************

        section gnippi,bss_C

bitplane:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256
        end

;***************************************************************************
```

In questo esempio vediamo un nuovo effetto ottenuto grazie ad una maschera. Disegnamo un'immagine sullo schermo una colonna di pixel per volta partendo da destra. In pratica dobbiamo copiare sullo schermo l'immagine una "colonna" di pixel alla volta. La larghezza minima di una blittata, pero' e' di una word, pari a 16 pixel. Quindi se eseguissimo semplici copie, potremmo solo copiare gruppi di 16 pixel. Fortunatamente, pero' ci sono le maschere. Facendo blittate larghe una word, entrambe le maschere si applicano alla word blittata. A noi comunque ne basta una, per esempio BLTALWM (sarebbe comunque la stessa cosa se usassimo BLTAFWM). Il trucco e' questo:
copiamo la stessa colonna di 1 word 16 volte, ogni volta nella stessa posizione dello schermo e ogni volta cambiamo la maschera in modo da far vedere una colonna di pixel in piu'.
In pratica, la prima volta facciamo la copia con la maschera al valore %1000000000000000 in modo da far vedere solo la prima colonna di pixel. La seconda blittata avviene nella stessa posizione della precedente, quindi sovrascrive quello che abbiamo disegnato la prima volta. Come maschera usiamo il valore %1100000000000000, in modo che si vedranno solo le prime 2 colonne di pixel. La terza volta usiamo come maschera %1110000000000000 e disegnamo le prime 3 colonne di pixel, e cosi' via. La 16-esima volta usiamo come maschera %1111111111111111 in modo da disegnare tutte e 16 le colonne di pixel che compongono la colonna di word. A questo punto dobbiamo iniziare a disegnare la prima colonna di pixel della colonna di word seguente, pertanto ci spostiamo di una word a destra, sia nella sorgente che nella destinazione e ricominciamo con la maschera al valore %1000000000000000. Poiche' le nostre blittate sono larghe solo una word, non sovrascriveremo la colonna di word precedente che pertanto rimane disegnata.
Notate il modo in cui sono ottenuti i valori della maschera. All'inizio si mette il valore di partenza (%1000000000000000) nel registro D0. Tale registro viene copiato in BLTALWM, in modo che la prima blittata faccia passare solo la prima colonna di pixel. Dopo la blittata viene eseguita una ASR #1,D0. Questa istruzione, come sapete, effettua uno shift verso destra del contenuto del registro D0. Inoltre (a differenza della LSR) conserva il segno, cioe' fa entrare da sinistra un bit dello stesso valore di quello che era il bit piu' a sinistra (cioe' il bit di segno) prima della shiftata. In questo caso, il bit di segno vale 1, pertanto da sinistra entra un'altro 1. In questo modo

```
il registro D0 assume il valore %1100000000000000. Questo valore viene usato
come maschera per la seconda blittata, e poi viene eseguita un'altra ASR
che porta D0 al valore %1110000000000000. Questo meccanismo viene ripetuto
ad ogni iterazione, generando tutti i valori della maschera.
Per ulteriori chiarimenti sulla ASR consultate la lezione 68000-2.TXT.
```

Lezione9h3r

```
; Lezione9h3r.s Facciamo apparire un immagine una colonna di pixel alla volta
;               Tasto destro per eseguire la blittata, sinistro per uscire.
;               Nota: immagine in RAWBLIT (o interleaved, se preferite).

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   Mostra          ; esegui la routine

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts
```

```
; ********************** LA ROUTINE CHE MOSTRA LA FIGURA ******************

Mostra:

; valori iniziali dei puntatori

        lea     picture,a0              ; punta all'inizio della figura
        lea     bitplane,a1             ; punta all'inizio del primo bitplane

        moveq   #20-1,d7                ; esegui per ogni "colonna" di word.
                                        ; lo schermo e' largo 20 word, quindi
                                        ; ci sono 20 colonne.

FaiTutteLeWord:
        moveq   #16-1,d6                ; 16 pixel per ogni word.
        move.w  #%1000000000000000,d0   ; valore della maschera all'inizio del
                                        ; loop interno. Fa passare solo il
                                        ; pixel piu' a sinistra della word.
FaiUnaWord:

; aspetta il vblank in modo da disegnare una colonna di pixel ad ogni
; fotogramma.

WaitWblank:
        CMP.b   #$ff,$dff006            ; vhposr - aspetta la linea 255
        bne.s   WaitWblank
Aspetta:
        CMP.b   #$ff,$dff006            ; vhposr - ancora linea 255 ?
        beq.s   Aspetta

        btst    #6,2(a5)        ; dmaconr -aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
        move.w  #$ffff,$44(a5)          ; BLTAFWM - fa passare tutti i bit
        move.w  d0,$46(a5)              ; Carica il valore della maschera nel
                                        ; registro BLTALWM
; carica i puntatori

        move.l  a0,$50(a5)              ; bltapt
        move.l  a1,$54(a5)              ; bltdpt

; Sia per la sorgente che per la destinazione blittiamo una word appartenente
; ad uno schermo largo 20 word. Il modulo quindi vale 2*(20-1)=38=$26.
; Poiche' i 2 registri hanno indirizzi consecutivi, si puo' usare una sola
; istruzione invece che 2:

        move.l #$00260026,$64(a5)       ; bltamod e bltdmod

; blittiamo una "colonna" di word alta 256 righe (tutto lo schermo)

        move.w  #(3*256*64)+1,$58(a5)   ; bltsize
                                        ; altezza 256 linee di 3 planes
                                        ; larghezza 1 word

        asr.w   #1,d0                   ; calcola la maschera per la prossima
                                        ; blittata. Fa passare ogni volta un
                                        ; bit in piu' rispetto alla volta
                                        ; precedente.
```

```
        dbra    d6,FaiUnaWord           ; ripeti per tutti i pixel

        addq.w  #2,a0                   ; punta alla word seguente
        addq.w  #2,a1                   ; punta alla word seguente

        dbra    d7,FaiTutteLeWord       ; ripeti per tutte le word

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

PICTURE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                ; qua carichiamo la figura in
                                ; formato RAWBLIT (o interleaved),
                                ; convertita col KEFCON.

;*****************************************************************************

        section gnippi,bss_C
```

```
bitplane:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256
        end

;*****************************************************************************

Questo esempio e' la versione rawblit di lezione9h3.s.
Confrontate le differenze nelle formule per il calcolo dei valori da scrivere
nei registri del blitter.
```

Lezione9h4

```
; Lezione9h4.s  Sparizione tramite scorrimento verso destra di un immagine
;               tramite shift + maschera BLTALWM.
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   Scorri          ; esegui la routine di scorrimento

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
```

```
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


;******************************************************************************
; Questa routine fa scomparire progressivamente un immagine
; facendola scorrere verso destra
;******************************************************************************

Scorri:
        move.w  #160-1,d7       ; Il loop va eseguito una volta per ogni pixel
                                ; l'immagine e' larga 160 pixel (10 words)

; In questo esempio copiamo un'immagine su se stessa, ma shiftandola
; continuamente di un pixel, in modo da farla scorrere.
; Pertanto gli indirizzi sorgente e destinazione sono uguali

ScorriLoop:

; Aspetta il vblank in modo da far scorrere l'immagine di un pixel ad ogni
; fotogramma.

WaitWblank:
        CMP.b   #$ff,$dff006            ; vhposr - aspetta la linea 255
        bne.s   WaitWblank
Aspetta:
        CMP.b   #$ff,$dff006            ; vhposr - ancora linea 255 ?
        beq.s   Aspetta

        move.l  #bitplane+((20*50)+64/16)*2,d0          ; ind. sorgente e
                                                        ; destinazione

        moveq   #3-1,d5                 ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$19f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l  #$fffffffe,$44(a5)      ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $fffe = %1111111111111110
                                        ;           cancella l'ultimo bit

; carica i puntatori

        move.l  d0,$50(a5)                      ; bltapt - sorgente
        move.l  d0,$54(a5)                      ; bltdpt - destinazione

; il modulo e' calcolato come al solito

        move.l  #$00140014,$64(a5)              ; bltamod e bltdmod
        move.w  #(20*64)+160/16,$58(a5)         ; bltsize
                                                ; altezza 20 linee
                                                ; largo 160 pixel (= 10 words)

        add.l   #40*256,d0                      ; punta al prossimo plane
        dbra    d5,PlaneLoop
```

```
        dbra    d7,ScorriLoop                   ; ripeti per ogni pixel

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO = 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.raw"
                                ; qua carichiamo la figura in
                                ; formato RAWBLIT (o interleaved),
                                ; convertita col KEFCON.
        end

;*****************************************************************************
```

In questo esempio vediamo un nuovo effetto. Faremo sparire un'immagine dallo schermo facendola scorrere verso destra e cancellando i pixel che arrivano ad una certa posizione orizzontale. Questo effetto e' ottenuto mediante il blitter combinando insieme shifting e masking. Lo scorrimento verso destra e' ottenuto naturalmente mediante lo shift. L'immagine viene letta dallo schermo (non e' memorizzata in un altro buffer) attraverso il canale A, viene shiftata di un pixel e riscritta nello schermo nella stessa posizione. La sorgente e la destinazione coincidono perfettamente. La maschera della prima word fa passare tutti i bit. La maschera dell'ultima word, invece, assume il valore %1111111111111110 e quindi cancella il bit piu' a destra.

```
Se non usassimo questo accorgimento i pixel che escono dalla word piu‘ a destra
rientrerebbero nella word piu‘ a sinistra una riga piu‘ in basso (ne abbiamo
parlato durante la spiegazine dello shift). Poiche‘ stiamo usando uno schermo
interleaved, la riga piu‘ in basso appartiene ad un diverso plane, e
se i pixel si spostano da un plane all’altro succede un macello.
Provate a rendervene conto mettendo BLTALWM al valore $ffff.
Grazie alla maschera questo non avviene, perche‘ la maschera e‘ applicata al
dato letto PRIMA di fare lo shift.
Quindi il bit che dovrebbe uscire da destra viene azzerato dalla mask.
Una blittata fatta in questo modo fa scorrere la figura a destra di un pixel.
Ripetendo la blittata tante volte quanti sono i pixel che compongono
l’immagine in larghezza (nel nostro caso 160) otteniamo una sparizione completa
dell’immagine.
E‘ possibile far scorrere l’immagine piu‘ velocemente utilizzando un valore
di shift maggiore di 1. In questo caso pero‘ si deve modificare anche la
maschera in modo che cancelli tutti i pixel che lo shift farebbe uscire.
Per esempio usando uno shift di 4 pixel, la maschera deve cancellare i 4 pixel
piu‘ a destra, perche‘ altrimenti uscirebbero fuori.
Inoltre poiche‘ l’immagine scorre piu‘ velocemente, e‘ necessario ripetere
meno volte la routine per farla scomparire tutta.
Nel caso di uno shift di 4 pixel, sono necessarie 160/4=40 iterazioni
della routine.
Provate voi a modificare la velocita‘, provando altri valori di shift,
per esempio 2,8 o 3.
```

Lezione9h4r

```
; Lezione9h4r.s Sparizione tramite scorrimento verso destra di un immagine
;               (in formato rawblit) tramite shift + maschera BLTALWM.
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C’E‘ LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
```

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   Scorri          ; esegui la routine di scorrimento

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


;***************************************************************************
; Questa routine fa scomparire progressivamente un immagine
; facendola scorrere verso destra
;***************************************************************************

Scorri:
        move.w  #160-1,d7       ; Il loop va eseguito una volta per ogni pixel
                                ; l'immagine e' larga 160 pixel (10 words)

; In questo esempio copiamo un'immagine su se stessa, ma shiftandola
; continuamente di un pixel, in modo da farla scorrere.
; Pertanto gli indirizzi sorgente e destinazione sono uguali

        move.l  #bitplane+((20*3*50)+64/16)*2,d0        ; ind. sorgente e
                                                        ; destinazione

ScorriLoop:

; Aspetta il vblank in modo da far scorrere l'immagine di un pixel ad ogni
; fotogramma.

WaitWblank:
        CMP.b   #$ff,$dff006            ; aspetta la linea 255
        bne.s   WaitWblank
Aspetta:
        CMP.b   #$ff,$dff006            ; ancora linea 255 ?
        beq.s   Aspetta

        btst    #6,2(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$19f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l  #$fffffffe,$44(a5)      ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $fffe = %1111111111111110
                                        ;               cancella l'ultimo bit

; carica i puntatori
```

```
        move.l  d0,$50(a5)                      ; bltapt - sorgente
        move.l  d0,$54(a5)                      ; bltdpt - destinazione

; il modulo e‘ calcolato come al solito

        move.l  #$00140014,$64(a5)              ; bltamod e bltdmod
        move.w  #(3*20*64)+160/16,$58(a5)       ; bltsize
                                                ; altezza 20 linee e 3 planes
                                                ; largo 160 pixel (= 10 words)

        dbra    d7,ScorriLoop                   ; ripeti per ogni pixel

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C’E‘ UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.
        end
```

```
;*****************************************************************************

Questo esempio e' la versione rawblit di lezione9h4.s.
Confrontate le differenze nelle formule per il calcolo dei valori da scrivere
nei registri del blitter.
```

25.9 Lezione9i1

```
; Lezione9i1.s  Di nuovo il pesce! Ma ancora piu' furbo!

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        moveq   #0,d1                   ; coordinata orizzontale a 0
        move.w  #(320-32)-1,d7          ; muove per 320 pixel MENO la larghezza
                                        ; reale del BOB, per fare in modo che
                                        ; il suo primo pixel a sinistra si
                                        ; fermi quando quello a destra arriva
                                        ; alla fine dello schermo.
Loop:
        cmp.b   #$ff,$6(a5)     ; VHPOSR - aspetta la linea $ff
        bne.s   loop
Aspetta:
        cmp.b   #$ff,$6(a5)     ; ancora linea $ff?
        beq.s   Aspetta

        lea     bitplane,a0     ; destinazione in a0
        move.w  d1,d0
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
```

```
        move.w  d1,d2
        lsr.w   #3,d2           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d2       ; escludo il bit 0
        add.w   d2,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione
        addq.w  #1,d1           ; aggiungi 1 alla coordinata orizzontale

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1


; Ora, come spiegato nella teoria, cogliamo l' occasione di scrivere i valori
; in registri ADIACENTI con un singolo 'move.l'

        move.l  #$01000000,$40(a5)      ; BLTCON0 + BLTCON1
        move.w  #$0000,$66(a5)          ; BLTDMOD
        move.l  #bitplane,$54(a5)       ; BLTDPT
        move.w  #(64*256)+20,$58(a5)    ; provate a togliere questa linea
                                        ; e lo schermo non verra' pulito,
                                        ; dunque il pesce lascera' la "scia"

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$fffe0024,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-4=36=$24 come al solito
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*6)+2,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; blittiamo 2 word, la seconda delle
                                        ; quali viene azzerata dalla maschera
                                        ; per permettere lo shift

        btst    #6,$bfe001              ; mouse premuto?
        beq.s   quit

        dbra    d7,loop

Quit:
        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
```

```
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; BplCon0 - 1 bitplane LowRes

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Il pesciolino: stavolta memorizziamo solo le word che ci interessano
; la word nulla viene creata dalla maschera.

Figura:
        dc.w    %1000001111100000
        dc.w    %1100111111111000
        dc.w    %1111111111101100
        dc.w    %1111111111111110
        dc.w    %1100111111111000
        dc.w    %1000001111100000

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

Questo esempio e‘ una modifica dell’esempio lezione9e3.s che, come abbiamo spiegato nella lezione, ci consente di evitare lo spreco di memoria della colonna di word in piu‘. Infatti la nostra figura e‘ larga appena una word. Ciononostante eseguiamo delle blittate larghe 2 words. Per azzerare la seconda word (cioe‘ l’ultima della riga) settiamo BLTALWM al valore $0000. Abbiamo poi il problema che il puntatore della sorgente (canale A) si sposta di una word di troppo. Per farlo tornare indietro settiamo il modulo della sorgente (BLTAMOD) al valore $fffe=-2.

Lezione9i2

```
; Lezione9i2.s  BOB a colori
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        bsr.w   LeggiMouse              ; Leggi coordinate
        bsr.s   ControllaCoordinate     ; evita che il bob esca dallo schermo
        bsr.s   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   CancellaOggetto         ; cancella il bob dalla vecchia
                                        ; posizione

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:
        rts

;*****************************************************************************
; Questa routine fa in modo che le coordinate del bob rimangano sempre
; all'interno dello schermo.
;*****************************************************************************

ControllaCoordinate:
        tst.w   ogg_x           ; controlla X
        bpl.s   NoMinX          ; controlla bordo sinistro
        clr.w   ogg_x           ; se X e' negativa, pone X=0
        bra.s   controllaY      ; va a controllare la Y

NoMinX:
```

```
        cmp.w   #319-32,ogg_x   ; Controlla il bordo destro. In X_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; sinistro del bob. Se esso ha raggiunto
                                ; 319-32, allora il bordo destro ha raggiunto
                                ; la coordinata 319.
        bls.s   controllaY      ; Se e' minore tutto bene, controlla la Y
        move.w  #319-32,ogg_x   ; altrimenti fissa la coordinata sul bordo.

controllaY:
        tst.w   ogg_y           ; controlla bordo in alto
        bpl.s   NoMinY          ; se e' positiva controlla in basso
        clr.w   ogg_y           ; altrimenti poni Y=0
        bra.s   EndControlla    ; ed esci

NoMinY:
        cmp.w   #255-11,ogg_y   ; controlla il bordo basso. In Y_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; alto del bob. Se esso ha raggiunto
                                ; Y=255-11, allora il bordo basso ha raggiunto
                                ; la coordinata Y=255
        bls.s   EndControlla    ; se e' minore tutto bene, controlla la Y
        move.w  #255-11,ogg_y   ; altrimenti fissa la coordinata sul bordo.
EndControlla:
        rts

;*****************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato normale, e pertanto
; sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
; tecnica di mascherare l'ultima word del BOB vista nella lezione.
;*****************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
```

```
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito
        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     4*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni plane e' largo 2 words e alto
                                        ; 11 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

;***************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob alto 6 linee e largo 3 words
;***************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0022,$66(a5)          ; BLTDMOD=40-6=34=$22
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
                                        ; il BOB
```

```
        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop2

        rts

*****************************************************************************
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili OGG_X e OGG_Y
;*****************************************************************************

LeggiMouse:
        move.b  $dff00a,d1      ; JOYODAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_y        ; modifica posizione oggetto

no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_x        ; modifica pos. oggetto
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
```

```
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 32 pixel (2 words)
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000       ; plane 1
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $0ffffe00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000

        dc.l    $00000000       ; plane 2
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000
        dc.l    $00000000

        dc.l    $007fc000       ; plane 3
        dc.l    $03803800
        dc.l    $04000400
        dc.l    $081f8200
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $081f8200
        dc.l    $04000400
        dc.l    $03803800
        dc.l    $007fc000

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.raw"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.
        end

;***************************************************************************

In questo esempio abbiamo un BOB a colori che si muove sullo schermo comandato
```

dal mouse. Sullo schermo c'e' un' immagine di sfondo. Viene usato il formato interleaved. Per il BOB usiamo la tecnica di mascherare l'ultima word che abbiamo gia' usato nell'esempio lezione9i1.s. Il programma si compone di 4 routine che vengono chiamate ciclicamente. La routine "LeggiMouse" e' la stessa usata nella lezione7. Legge dal mouse le cordinate del BOB e le memorizza nelle variabili X_OGG e Y_OGG. La routine "ControllaCoordinate" serve ad impedire che il BOB esca dallo schermo. La routine "DisegnaOggetto" e' la routine che disegna il BOB sullo schermo, ed e' analoga a quelle gia' viste prima. Notate che questa routine quando disegna sovrascrive lo sfondo. Una volta disegnato il BOB il programma attende il prossimo vertical blank, per permettere la visualizzazione dell'immagine con il BOB nella posizione corrente. Dopo il vertical blank viene chiamata la routine "CancellaOggetto" che cancella il BOB dalla posizione attuale. Questa routine effettua una cancellazione con il blitter del rettangolo che contiene il BOB. Al posto del BOB appare spazio vuoto. Successivamente ricomincia il loop principale, con la lettura delle nuove coordinate, il controllo e il disegno del BOB nella nuova posizione. Il grande "difetto" di questo programma e' che il BOB cancella le parti di sfondo sopra le quali passa.

Lezione9i2r

```
; Lezione9i2r.s BOB a colori (interleaved)
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA PRIMA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA
```

```
mouse:
        bsr.w   LeggiMouse              ; Leggi coordinate
        bsr.s   ControllaCoordinate     ; evita che il bob esca dallo schermo
        bsr.s   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   CancellaOggetto         ; cancella il bob dalla vecchia
                                        ; posizione

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:
        rts

;*****************************************************************************
; Questa routine fa in modo che le coordinate del bob rimangano sempre
; all'interno dello schermo.
;*****************************************************************************

ControllaCoordinate:
        tst.w   ogg_x           ; controlla X
        bpl.s   NoMinX          ; controlla bordo sinistro
        clr.w   ogg_x           ; se X e' negativa, pone X=0
        bra.s   controllaY      ; va a controllare la Y

NoMinX:
        cmp.w   #319-32,ogg_x   ; Controlla il bordo destro. In X_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; sinistro del bob. Se esso ha raggiunto
                                ; 319-32, allora il bordo destro ha raggiunto
                                ; la coordinata 319.
        bls.s   controllaY      ; Se e' minore tutto bene, controlla la Y
        move.w  #319-32,ogg_x   ; altrimenti fissa la coordinata sul bordo.

controllaY:
        tst.w   ogg_y           ; controlla bordo in alto
        bpl.s   NoMinY          ; se e' positiva controlla in basso
        clr.w   ogg_y           ; altrimenti poni Y=0
        bra.s   EndControlla    ; ed esci

NoMinY:
        cmp.w   #255-11,ogg_y   ; controlla il bordo basso. In Y_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; alto del bob. Se esso ha raggiunto
                                ; Y=255-11, allora il bordo basso ha raggiunto
                                ; la coordinata Y=255
        bls.s   EndControlla    ; se e' minore tutto bene, controlla la Y
        move.w  #255-11,ogg_y   ; altrimenti fissa la coordinata sul bordo.
EndControlla:
        rts

;*****************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato interleaved, e pertanto
; sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
```

```
; tecnica di mascherare l'ultima word del BOB vista nella lezione.
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e' costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)        ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)
        rts

;***************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob alto 6 linee e largo 3 words
;***************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e' costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
```

```
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0022,$66(a5)          ; BLTDMOD=40-6=34=$22
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
        rts                             ; il BOB

*****************************************************************************
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili OGG_X e OGG_Y
;*****************************************************************************

LeggiMouse:
        move.b  $dff00a,d1      ; JOY0DAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e' fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_y        ; modifica posizione oggetto

no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e' fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_x        ; modifica pos. oggetto
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
```

```
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' LA SECONDA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato interleaved, largo 32 pixel (2 words)
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000,$00000000,$007fc000
        dc.l    $03fff800,$007fc000,$03803800
        dc.l    $07fffc00,$03fff800,$04000400
        dc.l    $0fffe00,$07fffc00,$081f8200
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $0fffe00,$07fffc00,$081f8200
        dc.l    $07fffc00,$03fff800,$04000400
        dc.l    $03fff800,$007fc000,$03803800
        dc.l    $007fc000,$00000000,$007fc000

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.
        end

;***************************************************************************

Questo esempio e' la versione rawblit di lezione9i2.s.
Confrontate le differenze nelle formule per il calcolo dei valori da scrivere
nei registri del blitter.
```

Figura 25.8: Lezione 9i2r

Lezione9i3

```
; Lezione9i3.s  BOB con ripristino dello sfondo.
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
```

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        bsr.w   LeggiMouse              ; leggi coordinate
        bsr.s   ControllaCoordinate     ; evita che il bob esca dallo schermo
        bsr.w   SalvaSfondo             ; salva lo sfondo
        bsr.s   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   RipristinaSfondo        ; ripristina lo sfondo

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts


;*****************************************************************************
; Questa routine fa in modo che le coordinate del bob rimangano sempre
; all'interno dello schermo.
;*****************************************************************************

ControllaCoordinate:
        tst.w   ogg_x           ; controlla X
        bpl.s   NoMinX          ; controlla bordo sinistro
        clr.w   ogg_x           ; se X e' negativa, pone X=0
        bra.s   controllaY      ; va a controllare la Y

NoMinX:
        cmp.w   #319-32,ogg_x   ; controlla il bordo destro. In X_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; sinistro del bob. Se esso ha raggiunto
                                ; 319-32, allora il bordo destro ha raggiunto
                                ; la coordinata 319
        bls.s   controllaY      ; se e' minore tutto bene, controlla la Y
        move.w  #319-32,ogg_x   ; altrimenti fissa la coordinata sul bordo.

controllaY:
        tst.w   ogg_y           ; controlla bordo in alto
        bpl.s   NoMinY          ; se e' positiva controlla in basso
        clr.w   ogg_y           ; altrimenti poni Y=0
        bra.s   EndControlla    ; ed esci

NoMinY:
        cmp.w   #255-11,ogg_y   ; controlla il bordo basso. In Y_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; alto del bob. Se esso ha raggiunto
                                ; Y=255-11, allora il bordo basso ha raggiunto
                                ; la coordinata Y=255
```

```
        bls.s   EndControlla    ; se e' minore tutto bene, controlla la Y
        move.w  #255-11,ogg_y   ; altrimenti fissa la coordinata sul bordo.
EndControlla:
        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato normale, e pertanto
; sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
; tecnica di mascherare l'ultima word del BOB vista nella lezione
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito
        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     4*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni plane e' largo 2 words e alto
                                        ; 11 righe
```

```
        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

;***************************************************************************
; Questa routine copia il rettangolo di sfondo che verra' sovrascritto dal
; BOB in un buffer
;***************************************************************************

SalvaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     Buffer,a1       ; indirizzo destinazione
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00220000,$64(a5)      ; BLTAMOD=40-4=36=$24
                                        ; BLTDMOD=0 nel buffer
        move.l  a0,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  a1,$54(a5)              ; BLTDPT - buffer
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     40*256(a0),a0           ; punta al prossimo plane sorgente
        lea     6*11(a1),a1             ; punta al prossimo plane destinazione
                                        ; ogni blittata e' larga 3 words e alto
                                        ; 11 righe
        dbra    d7,PlaneLoop2

        rts

;***************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione.
;***************************************************************************

RipristinaSfondo:
```

```
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        lea     Buffer,a1       ; indirizzo sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop3:
        btst    #6,2(a5)         ; dmaconr
WBlit4:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00000022,$64(a5)      ; BLTAMOD=0 (buffer)
                                        ; BLTDMOD=40-6=34=$22
        move.l  a1,$50(a5)              ; BLTAPT (buffer)
        move.l  a0,$54(a5)              ; BLTDPT (schermo)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        lea     6*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni blittata e‘ larga 3 words e alto
                                        ; 11 righe
        dbra    d7,PlaneLoop3
        rts

;***************************************************************************
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili OGG_X e OGG_Y
;***************************************************************************

LeggiMouse:
        move.b  $dff00a,d1      ; JOY0DAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_y        ; modifica posizione oggetto

no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
```

```
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_x        ; modifica pos. oggetto
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO = 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e‘ in formato normale, largo 32 pixel (2 words)
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000       ; plane 1
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $0ffffe00
        dc.l    $07fffc00
        dc.l    $03fff800
```

```
        dc.l    $007fc000

        dc.l    $00000000       ; plane 2
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000
        dc.l    $00000000

        dc.l    $007fc000       ; plane 3
        dc.l    $03803800
        dc.l    $04000400
        dc.l    $081f8200
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $081f8200
        dc.l    $04000400
        dc.l    $03803800
        dc.l    $007fc000

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.raw"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.

;***************************************************************************

        SECTION BUFFER,BSS_C

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 11, larghezza 3 words
; 3 bitplanes

Buffer:
        ds.w    11*3*3

        end

;***************************************************************************
```

In questo esempio affrontiamo il problema dello sfondo con i BOB.
Non forniremo ancora la soluzione definitiva che richiede la comprensione di caratteristiche del blitter ancora non spiegate nel corso. Comunque faremo un primo passo. L'idea e' la seguente: prima di disegnare il BOB sullo schermo copiamo la parte di sfondo che verrebbe sovrascritta dal BOB in un buffer. Poi disegnamo normalmente il BOB. Dopo il vertical blank dobbiamo cancellare il BOB prima di ridisegnarlo nella nuova posizione. Invece di cancellare semplicemente (cosa che lascerebbe il pezzo di schermo vuoto) ci ricopiamo sopra lo sfondo che c'era prima. In questo modo cancelliamo la vecchia copia del BOB e rimettiamo lo sfondo che c'era prima del suo passaggio.
Il problema di questa tecnica, come potete vedere, e' che non si vede lo sfondo nelle zone nel rettangolo che racchiude il BOB. Questo e' dovuto al fatto che le parti del BOB colorate con il colore 0 non sono trasparenti come

```
accadeva per gli sprite, ma rappresentano il colore di sfondo.
Vedremo successivamente la soluzione a questo problema.
```

Lezione9i3r

```
; Lezione9i3r.s BOB rawblit con ripristino dello sfondo.
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA PRIMA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        bsr.w   LeggiMouse              ; leggi coordinate
        bsr.s   ControllaCoordinate     ; evita che il bob esca dallo schermo
        bsr.w   SalvaSfondo             ; salva lo sfondo
        bsr.s   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   RipristinaSfondo        ; ripristina lo sfondo
```

```
        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts


;*****************************************************************************
; Questa routine fa in modo che le coordinate del bob rimangano sempre
; all'interno dello schermo.
;*****************************************************************************

ControllaCoordinate:
        tst.w   ogg_x           ; controlla X
        bpl.s   NoMinX          ; controlla bordo sinistro
        clr.w   ogg_x           ; se X e' negativa, pone X=0
        bra.s   controllaY      ; va a controllare la Y

NoMinX:
        cmp.w   #319-32,ogg_x   ; controlla il bordo destro. In X_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; sinistro del bob. Se esso ha raggiunto
                                ; 319-32, allora il bordo destro ha raggiunto
                                ; la coordinata 319
        bls.s   controllaY      ; se e' minore tutto bene, controlla la Y
        move.w  #319-32,ogg_x   ; altrimenti fissa la coordinata sul bordo.

controllaY:
        tst.w   ogg_y           ; controlla bordo in alto
        bpl.s   NoMinY          ; se e' positiva controlla in basso
        clr.w   ogg_y           ; altrimenti poni Y=0
        bra.s   EndControlla    ; ed esci

NoMinY:
        cmp.w   #255-11,ogg_y   ; controlla il bordo basso. In Y_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; alto del bob. Se esso ha raggiunto
                                ; Y=255-11, allora il bordo basso ha raggiunto
                                ; la coordinata Y=255
        bls.s   EndControlla    ; se e' minore tutto bene, controlla la Y
        move.w  #255-11,ogg_y   ; altrimenti fissa la coordinata sul bordo.
EndControlla:
        rts

;*****************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato interleaved, e pertanto
; sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
; tecnica di mascherare l'ultima word del BOB vista nella lezione
;*****************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e' costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
```

```
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito
        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)

        rts

;***************************************************************************
; Questa routine copia il rettangolo di sfondo che verra' sovrascritto dal
; BOB in un buffer
;***************************************************************************

SalvaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e' costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione
        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
```

```
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00220000,$64(a5)      ; BLTAMOD=40-4=36=$24
                                        ; BLTDMOD=0 nel buffer
        move.l  a0,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  #Buffer,$54(a5)         ; BLTDPT - buffer
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)
        rts

;***************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione.
;***************************************************************************

RipristinaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e' costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5)         ; dmaconr
WBlit4:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00000022,$64(a5)      ; BLTAMOD=0 (buffer)
                                        ; BLTDMOD=40-6=34=$22
        move.l  #Buffer,$50(a5)         ; BLTAPT (buffer)
        move.l  a0,$54(a5)              ; BLTDPT (schermo)
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)
        rts

;***************************************************************************
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili OGG_X e OGG_Y
;***************************************************************************

LeggiMouse:
        move.b  $dff00a,d1      ; JOY0DAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e' fermo
```

```
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_y        ; modifica posizione oggetto

no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e' fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_x        ; modifica pos. oggetto
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' LA SECONDA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato interleaved, largo 32 pixel (2 words)
```

```
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000,$00000000,$007fc000
        dc.l    $03fff800,$007fc000,$03803800
        dc.l    $07fffc00,$03fff800,$04000400
        dc.l    $0ffffe00,$07fffc00,$081f8200
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $0ffffe00,$07fffc00,$081f8200
        dc.l    $07fffc00,$03fff800,$04000400
        dc.l    $03fff800,$007fc000,$03803800
        dc.l    $007fc000,$00000000,$007fc000

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.

;***************************************************************************

        SECTION BUFFER,BSS_C

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 11, larghezza 3 words
; 3 bitplanes

Buffer:
        ds.w    11*3*3

        end

;***************************************************************************

Questo esempio e' la versione rawblit di lezione9i3.s.
Confrontate le differenze nelle formule per il calcolo dei valori da scrivere
nei registri del blitter.
```

Lezione9i4

```
; Lezione9i4.s  BOB con sfondo "finto"
;               Tasto sinistro per uscire.

        SECTION bau,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


; costanti bordi.
```

```
Lowest_Floor    equ     200     ; bordo in basso
Right_Side      equ     287     ; bordo a destra


START:

; puntiamo i bitplanes
        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

;       Puntiamo il quarto bitplane (lo sfondo)

        LEA     BPLPOINTERS,A0          ; puntatori COP
        move.l  #SfondoFinto,d0         ; indirizzo sfondo
        move.w  d0,30(a0)               ; lo sfondo e‘ il bitplane 4
        swap    d0
        move.w  d0,26(a0)               ; scrivi word alta

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        bsr.s   MuoviOggetto            ; muove il bob
        bsr.w   DisegnaOggetto          ; disegna il bob


        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   CancellaOggetto         ; cancella il bob dalla vecchia
                                        ; posizione

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts


;*****************************************************************************
; Questa routine muove il bob controllando che non superi i bordi
;*****************************************************************************

MuoviOggetto:
```

```
        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  vel_x(pc),d2            ; dx (velocita‘ X)
        move.w  vel_y(pc),d3            ; dy (velocita‘ Y)
        add.w   d2,d0                   ; x = x + dx
        add.w   d3,d1                   ; y = y + dy
        addq.w  #1,d3                   ; aggiunge la gravita‘
                                        ; (aumenta la velocita‘)
        cmp.w   #Lowest_Floor,d1        ; controlla bordo in basso
        blt.s   UO_NoBounce1

        subq.w  #1,d3                   ; togli l’aumento di velocita‘
        neg.w   d3                      ; cambia il segno della velocita‘ dy
                                        ; invertendo la direzione del moto
        move.w  #Lowest_Floor,d1        ; riparti dal bordo
UO_NoBounce1:

        cmp.w   #Right_Side,d0          ; controlla bordo destro
        blt.s   UO_NoBounce2            ; se supera il bordo destro..
        sub.w   #Right_Side,d0          ; distanza dal bordo
        neg.w   d0                      ; inverti la distanza
        add.w   #Right_Side,d0          ; aggiungi coordinata bordo
        neg.w   d2                      ; inverti direzione del moto
UO_NoBounce2:
        btst    #15,d0                  ; controlla bordo sinistro (X=0)
        beq.s   UO_NoBounce3            ; se la X e‘ negativa...
        neg.w   d0                      ; .. fa il rimbalzo
        neg.w   d2                      ; inverti direzione del moto
UO_NoBounce3:
        move.w  d0,ogg_x                ; aggiorna posizione e velocita‘
        move.w  d1,ogg_y
        move.w  d2,vel_x
        move.w  d3,vel_y

        rts


;*****************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato normale (non interleaved)
; e sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
; tecnica di mascherare l’ultima word del BOB vista nella lezione
;*****************************************************************************

DisegnaOggetto:
        lea     BITPLANE1,a0    ; indirizzo bitplane
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga occupa 40 bytes

        add.l   d0,a0           ; aggiungi offset Y

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche’ vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
```

```
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.l   #$0000fffe,d1   ; escludo il bit 0 del
        add.l   d1,a0           ; aggiungi l'offset X, trovando l'indirizzo
                                ; della destinazione

        move.l  a0,IndirizzoOgg         ; memorizza l'indirizzo della
                                        ; destinazione per la routine
                                        ; di cancellazione

        lea     Ball_Bob,a1             ; puntatore alla figura
        moveq   #3-1,d7                 ; bitplane counter

DrawLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)
        bne.s   WBlit2

        move.w  d0,$40(a5)              ; BLTCON0 - scrivi valore di shift
        move.w  #$0000,$42(a5)          ; BLTCON1 - modo ascendente
        move.l  #$ffff0000,$44(a5)      ; BLTAFWM e BLTLWM
        move.w  #$FFFE,$64(a5)          ; BLTAMOD
        move.w  #40-6,$66(a5)           ; BLTDMOD
        move.l  a1,$50(a5)              ; BLTAPT - puntatore figura
        move.l  a0,$54(a5)              ; BLTDPT - puntatore bitplanes

        move.w  #(31*64)+3,$58(a5)      ; BLTSIZE - altezza 31 linee
                                        ; largh. 3 word (48 pixel).

        add.l   #4*31,a1                ; indirizzo prossimo plane immagine
        add.l   #40*256,a0              ; indirizzo prossimo plane destinazione

        dbra    d7,DrawLoop
        rts


;***************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob
;***************************************************************************

CancellaOggetto:
        moveq   #3-1,d7                 ; 3 bitplanes
        move.l  IndirizzoOgg(PC),a0     ; rileggi l'indirizzo destinazione

canc_loop:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move    #$0022,$66(a5)          ; BLTDMOD=40-6=34=$22
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*31)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
                                        ; il BOB

        add.l   #40*256,a0              ; indirizzo prossimo plane destinazione
        dbra    d7,canc_loop
```

```
        rts


; dati oggetto

IndirizzoOgg:
        dc.l    0       ; questa variabile contiene l'indirizzo della
                        ; destinazione

ogg_x:  dc.w    32              ; posizione X
ogg_y:  dc.w    50              ; posizione Y
vel_x:  dc.w    -3              ; velocita' X
vel_y:  dc.w    1               ; velocita' Y

;***************************************************************************

        SECTION MY_COPPER,CODE_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

        dc.w    $108,0          ; MODULO
        dc.w    $10a,0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000

        dc.w    $180,$000       ; color0 - sfondo
        dc.w    $190,$000

        dc.w    $182,$0A0                       ; colori da 1 a 7
        dc.w    $184,$040
        dc.w    $186,$050
        dc.w    $188,$061
        dc.w    $18A,$081
        dc.w    $18C,$020
        dc.w    $18E,$6F8

        dc.w    $192,$0A0                       ; colori da 9 a 15
        dc.w    $194,$040                       ; sono gli stessi valori
        dc.w    $196,$050                       ; caricati nei registri da 1 a 7
        dc.w    $198,$061
        dc.w    $19a,$081
        dc.w    $19c,$020
        dc.w    $19e,$6F8

        dc.w    $190,$345       ; colore 8 - pixel ad 1 dello sfondo

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $8007,$fffe     ; aspetta riga $80
        dc.w    $100,$4200      ; bplcon0 - 4 bitplanes lowres
                                ; attiva il bitplane 4 (sfondo)
```

```
; in questo spazio e' visualizzata la parte dello sfondo

        dc.w    $e007,$fffe     ; aspetta riga $e0
        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Figura Bob
Ball_Bob:
 DC.W $0000,$0000,$0000,$0000,$0000,$0000,$003F,$8000   ; plane 1
 DC.W $00C1,$E000,$017C,$E000,$02FE,$3000,$05FF,$5400
 DC.W $07FF,$1800,$0BFE,$AC00,$03FF,$1A00,$0BFE,$AC00
 DC.W $11FF,$1A00,$197D,$2C00,$0EAA,$1A00,$1454,$DC00
 DC.W $0E81,$3800,$0154,$F400,$02EB,$F000,$015F,$D000
 DC.W $00B5,$A000,$002A,$8000,$0000,$0000,$0000,$0000
 DC.W $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
 DC.W $0000,$0000,$0000,$0000,$0000,$0000

 DC.W $000F,$E000,$007F,$FC00,$01FF,$FF00,$03FF,$FF80   ; plane 2
 DC.W $07C1,$FFC0,$0F00,$FFE0,$1E00,$3FF0,$3C40,$5FF8
 DC.W $3CE0,$1FF8,$7840,$2FFC,$7800,$1FFC,$7800,$2FFC
 DC.W $F800,$1FFE,$F800,$2FFE,$FE00,$1FFE,$FC00,$DFFE
 DC.W $FE81,$3FFE,$FF54,$FFFE,$FFEB,$FFFE,$7FFF,$FFFC
 DC.W $7FFF,$FFFC,$7FFF,$FFFC,$3FFF,$FFF8,$3FFF,$FFF8
 DC.W $1FFF,$FFF0,$0FFF,$FFE0,$07FF,$FFC0,$03FF,$FF80
 DC.W $01FF,$FF00,$007F,$FC00,$000F,$E000

 DC.W $000F,$E000,$007F,$FC00,$01E0,$7F00,$0380,$0F80   ; plane 3
 DC.W $073E,$0AC0,$0CFF,$0560,$198F,$C2F0,$3347,$A0B8
 DC.W $32EB,$E158,$6647,$D0AC,$660B,$E05C,$4757,$D0AC
 DC.W $C7AF,$E05E,$A7FF,$D02E,$C1FF,$E05E,$A3FF,$202E
 DC.W $D17E,$C05E,$E0AB,$002E,$D014,$005E,$6800,$00AC
 DC.W $7000,$02DC,$7400,$057C,$2800,$0AF8,$3680,$55F8
 DC.W $1D54,$AAF0,$0EAB,$55E0,$0754,$ABC0,$03EB,$FF80
 DC.W $01FE,$FF00,$007F,$FC00,$000F,$E000

;*****************************************************************************

; Sfondo 320 * 100 1 Bitplane, raw normale.

SfondoFinto:
        incbin  "assembler2:sorgenti6/sfondo320*100.raw"

;*****************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256

        end

;*****************************************************************************
```

In questo esempio vedremo un bob che si muove su uno sfondo. L'effetto e‘
ottenuto pero‘ con un trucco che limita molto le prestazioni. Il trucco e‘
il seguente: usiamo 4 bitplanes, i primi 3 per disegnare il bob e il quarto

e' per lo sfondo. Lo sfondo e il bob hanno dunque piani separati.
Per far apparire il bob sopra lo sfondo, si fa in modo che il bitplane
dello sfondo non influenzi i colori del bob. Consideriamo per esempio un
pixel del bob formato prendendo plane 1=0, plane 2=1, plane 3=1.
Questo pixel muovendosi viene a trovarsi sovrapposto a tanti bit del plane 4.
Quando si trova in corrispondenza di un bit posto a 0, i 4 planes formeranno
la combinazione plane 1=0, plane 2=1, plane 3=1, plane 4=0 che definisce il
colore 6. Quando invece si trova in corrispondenza di un bit posto a 1, si
formera' la combinazione plane 1=0, plane 2=1, plane 3=1, plane 4=1 che
definisce il colore 14. Quindi i colori del bob cambiano a seconda della
zona di sfondo che attraversano. Noi vorremmo invece che il bob apparisse
sempre uguale passando sopra lo sfondo. Possiamo simulare questo effetto
in un modo molto semplice, rendendo uguali i colori contenuti nei registri
colore che differiscono solo per i bit dello sfondo. Tornando all'esempio, se
mettiamo lo stesso valore RGB sia nel registro COLOR06 che in COLOR14, quale
che sia il valore del bit del plane 4, il nostro pixel apparira' sempre dello
stesso colore. facendo lo stesso per tutti gli altri registri (cioe' ponendo
 COLOR01 = COLOR09, COLOR02=COLOR10, COLOR03=COLOR11 ecc) risolveremo
il problema. La parte "trasparente" del bob, e' quella che ha i 3 planes a 0,
che visualizza il colore 0 o il colore 8 a seconda del valore del bit nel plane
4. Tenendo diversi questi 2 colori, e' possibile visualizzare lo sfondo:
i bit a 0 dello sfondo appariranno del colore 0, mentre quelli a 1 appariranno
del colore 8. Per capire bene cosa succende, provate un po' a mettere nei
registri COLOR01-07 valori diversi da quelli di COLOR09-15: scoprirete subito
il trucco. Questa tecnica ha lo svantaggio di "sprecare" alcuni colori.
Infatti siamo costretti a scrivere valori RGB uguali in alcuni registri,
diminuendo il numero di colori visualizzabili. In questo esempio, utiliziamo 4
bitplanes, ma possiamo usare solo 8 colori per il bob e 2 per lo sfondo.
Sprechiamo dunque 6 colori. Se usassimo 3 planes per il bob e 2 per lo sfondo,
potremmo visualizzare 8+4=12 colori, contro i 32 normalmente permessi da 5
bitplanes. Come vedete dunque anche questa tecnica non e' l'ideale.
Ma non temete, prima o poi riusciremo a fare un bob come si deve!
Nel frattempo notate un po' di cose in questo listato:
1) Utiliziamo il trucco (gia' visto) della BLTLWM a 0 per risparmiare la
colonna di word a destra del bob;
2) Utiliziamo uno schermo NON interleaved per separare i planes di sfondo e
planes del bob.
3) Negli esempi precedenti calcoliamo l'indirizzo della destinazione del bob,
sia nella routine di disegno che in quella di cancellazione. In realta' tra
il disegno e la successiva cancellazione, il bob non cambia posizione (lo fa
solo DOPO la cancellazione) quindi il calcolo e' sempre lo stesso e lo si
potrebbe fare una sola volta. In questo esempio facciamo appunto questo:
il calcolo viene fatto nella routine DisegnaOggetto e viene memorizzato
nella variabile IndirizzoOgg. La routine di cancellazione si limita a rileggere
il risultato dalla variabile e ad usarlo.

Lezione9i5

```
; Lezione9i5.s  Clipping di Bob a destra. (By Erra Ugo)
;               Tasto sinistro per uscire.

        section CLippaD,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "Startup1.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
```

```
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


; Definiamo in queste equ le costanti relative al nostro bob...

XBob    equ     16*8    ; Dimenzione X del bob
YBob    equ     29      ; Dimenzione Y del bob
XWord   equ     8       ; Numero di word del bob

; Definiamo il limiti delo schermo

XMax    =       320-64          ; Limite orizontale destro dello schermo
XMin    =       0               ; Limite orizontale sinistro dello schermo
YMax    =       200-YBob        ; Limite verticale inferiore dello schermo
YMin    =       0               ; Limite verticale superiore dello schermo


Start:
        Lea     Screen,a0               ; prepariamo il puntatore
        Move.l  a0,d0                   ; al bitplane.
        Move.w  d0,BPLPointer1+6
        Swap    d0
        Move.w  d0,BPLPointer1+2

        Lea     $dff000,a6              ; CUSTOM REGISTER in a5
        Move.w  #DMASET,$96(a6)         ; DMACON - abilita bitplane, copper
        Move.l  #CopperList,$80(a6)     ; Puntiamo la nostra COP
        Move.w  d0,$88(a6)              ; Facciamo partire la COP
        Move.w  #0,$1fc(a6)             ; Disattiva l'AGA
        Move.w  #$c00,$106(a6)          ; Disattiva l'AGA
        Move.w  #$11,$10c(a6)           ; Disattiva l'AGA

        Moveq   #100,d0         ; d0 e' la coordinata x
        Move.w  #100,d1         ; d1 e' la coordinata y
        Moveq   #0,d2           ; azzeriamo il resto dei registri dati
        Moveq   #0,d3           ; bla bla bla
        Moveq   #0,d4           ; bla bla
        Moveq   #0,d5           ; bla
        Moveq   #0,d6
        Moveq   #0,d7

Loop:
        Cmpi.b  #$ff,$6(a6)
        Bne.s   Loop

        Bsr.w   LeggiJoyst      ; La routine legge lo stato del joystick
                                ; ed aggiorna x ed y direttamente nei registri
                                ; d0 e d1.
        Bsr.w   CheckLimit      ; Controlla se la routine e' nei limiti
        Bsr.w   CancellaSchermo ; pulisci lo schermo
        Bsr.s   ClipBobRight    ; clippa il bob lo piazza a video
        Btst    #6,$bfe001      ; Attende la pressione del tasto sinistro
        Bne.s   Loop            ; ...
        Rts

; ***************************************************************************
; La tecnica decritta viene implementata nel seguente modo:
; 1)Se la coordinata in alto a destra e uscita dal limite massimo allora
;   non blitta nulla.
; 2)Calcola di quanti pixel il bob e' uscito fuori, nel seguente modo
;   Xout=(x+xdim)-XMax
; 3)Si calcola quindi esattamente di quante word il bob è uscito fuori e
```

```
;   di quanti pixel, nel seguente modo XOut/16 e XOut mod 16.
; 4)A questo punto dalla tabella maskright preleviamo il valore
;   del registro BLTLWM tramite il valore XOut mod 16
; 5)Prepariamo il modulo A del blitter tramite l'operazione (XBob-XOut)/16
; ***************************************************************************

ClipBobRight:
        Movem.l d0-d7/a0,-(a7)
        Cmpi.w  #XMax,d0        ; Confronta la coordinata in alto a sinistra
                                ; con il XMax
        Bge.w   ExitClipRight   ; se e' maggiore allora il bob e' completamente
                                ; fuori, e quindi non facciamo nulla

        Move.w  #XBob,d7        ; d7=Dimensione del bob
        Add.w   d0,d7           ; Sommo a d7 la coordinata x, quindi d7 e'
                                ; uguale alla coordinata in alto a destra.
        Subi.w  #XMax,d7        ; Calcolo di quanti pixel il bob e'uscito fuori
        Ble.w   IsInLeft        ; Se il risultato e' mimore di zero allora
                                ; il bob e' uscito completamente fuori.

        Move.w  d7,d6           ; d7=d6=numero di pixel out
        Lsr.w   #4,d6           ; d6=d6/16 numero di word out
        Move.w  #XWord,d2       ; d2=numero di word del bob originariamente
        Andi.w  #15,d7          ; d7=numero di pixel out

                                ; Adesso calcolo il nuovo valore di bltsize
        Move.w  d2,d5           ; d5=numero di word del bob originariamente
        Sub.w   d6,d2           ; d2 numero di word in
        Move.w  #YBob,d3        ; Dimensione verticale in d3
        Lsl.w   #6,d3           ; Moltiplico d3 per 64
        Add.w   d2,d3           ; d3=bltsize ridotto

                                ; Calcoliamo il nuovo modulo della destinazione
        Moveq   #40,d4          ; Per calcolare il nuovo modulo della
                                ; destinazione. Non facciamo che sottrarre
                                ; le dimensioni restanti del bob a 40.
        Add.w   d5,d5           ; d5=d5*2 numero di byte del bob originariam.
        Add.w   d6,d6           ; d6=d6*2 modulo di A in byte
        Sub.w   d6,d5           ; d5=numero di byte out
        Sub.w   d5,d4           ; d4=modulo di D

        Moveq   #-1,d5
        Add.w   d7,d7           ; con d7 preleviamo il valore della maschera
        Lea     MaskRight,a0    ; in a0 l'ind. della tabella
        Move.w  (a0,d7.w),d5    ; d5=maskera

        Mulu    #40,d1          ; Da qui blitting normale...
        Move.w  d0,d2
        Lsr.w   #3,d0
        Add.w   d0,d1
        Lea     Screen,a0
        Adda.l  d1,a0
        Andi.w  #$000f,d2
        Ror.w   #4,d2           ; piu‘ efficente che fare LSL #4,d2 e
                                ; poi LSL #8,D2
        Ori.w   #$09f0,d2

        Btst    #6,2(a6)
WaitBlit1b:
        Btst    #6,2(a6)        ; dmaconr - aspetta che il blitter sia libero
        bne.s   WaitBlit1b
```

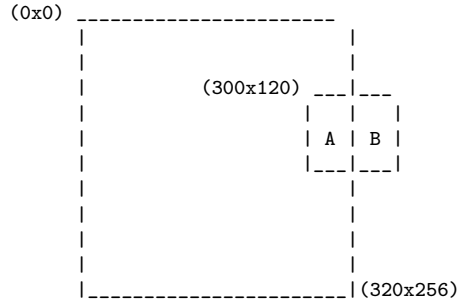
```
        Move.w  d2,$40(a6)      ; bltcon0
        Move.l  d5,$44(a6)      ; bltafwm
        Move.l  #Bob,$50(a6)    ; bltapt
        Move.l  a0,$54(a6)      ; bltdpt
        Move.w  d6,$64(a6)      ; bltamod
        Move.w  d4,$66(a6)      ; bltdmod
        Move.w  d3,$58(a6)      ; bltsize
        Movem.l (a7)+,d0-d7/a0
        Rts

IsInLeft:
        Mulu.w  #40,d1          ; In questo caso usufruiamo del blitter
        Move.w  d0,d2           ; normalmente poiche' il bob si trova entro
        Lsr.w   #3,d0           ; il limiti prefissati.
        Add.w   d0,d1
        Lea     Screen,a0
        Add.l   d1,a0
        Andi.w  #$000f,d2
        Ror.w   #4,d2
        Ori.w   #$09f0,d2

        Moveq   #-1,d7
        Clr.w   d7

        Btst    #6,2(a6)
WaitBlit1a:
        Btst    #6,2(a6)        ; dmaconr - aspetta che il blitter sia libero
        bne.s   WaitBlit1a

        Move.w  d2,$40(a6)      ; bltcon0
        Move.w  #0,$42(a6)      ; bltcon1
        Move.l  d7,$44(a6)      ; bltafwm
        Move.l  #Bob,$50(a6)    ; bltapt
        Move.l  a0,$54(a6)      ; bltdpt
        Move.w  #-2,$64(a6)     ; bltamod
        Move.w  #40-18,$66(a6)  ; bltdmod
        Move.w  #(29*64)+(144/16),$58(a6)       ; bltsize
ExitClipRight:
        Movem.l (a7)+,d0-d7/a0
        Rts

; ***************************************************************************
; Questa routine controlla che il bob non esca dai limti fisici dello
; schermo. Infatti abbiamo realizzato una routine che taglia le parti che
; escono al di fuori della nostra destra, ma non abbiamo fatto nulla per
; gli altri limiti dello schermo. Quindi questa routine controlla se le
; coordinate sono sempre nel range giusto.
; ***************************************************************************

CheckLimit:
        Cmpi.w  #XMin,d0        ; E' uscito dalla sinistra ?
        Bge.s   Limit2          ; no, allora vedi sopra e sotto
        Move.w  #XMin,d0        ; si, allora rimettilo nei nostri limiti
Limit2:
        Cmpi.w  #YMin,d1        ;E' uscito da sopra ?
        Bge.s   Limit3          ;no,allora vedi sotto
        Move.w  #YMin,d1        ;si, allora rimettilo nei limiti
        Bra.s   End_Limit       ;e' euindi esci fuori poiche' il nostro bob non
                                ;puo' stare contemporaneamente sopra e sotto.
Limit3:
        Cmpi.w  #YMax,d1        ; Come sopra ma controlliamo il limite
        Blt.s   End_Limit       ; verticale in basso.
```

```
        Move.w  #YMax,d1
End_Limit
        Rts



; ***************************************************************************
; Questa routine legge il joystick e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y
; ***************************************************************************

LeggiJoyst:
        Move.w  $dff00c,D3      ; JOY1DAT
        Btst.l  #1,D3           ; il bit 1 ci dice se si va a destra
        Beq.s   NODESTRA        ; se vale zero non si va a destra
        Addq.w  #1,d0           ; se vale 1 sposta a di un pixel lo sprite
        Bra.s   CHECK_Y         ; vai al controllo della Y
NODESTRA:
        Btst    #9,D3           ; il bit 9 ci dice se si va a sinistra
        Beq.s   CHECK_Y         ; se vale zero non si va a sinistra
        Subq.w  #1,d0           ; se vale 1 sposta lo sprite
CHECK_Y:
        Move.w  D3,D2           ; copia il valore del registro
        Lsr.w   #1,D2           ; fa scorrere i bit di un posto verso destra
        Eor.w   D2,D3           ; esegue l'or esclusivo. Ora possiamo testare
        Btst    #8,D3           ; testiamo se va in alto
        Beq.s   NOALTO          ; se no controlla se va in basso
        Subq.w  #1,d1           ; se si sposta lo sprite
        Bra.s   ENDJOYST
NOALTO:
        Btst    #0,D3           ; testiamo se va in basso
        Beq.s   ENDJOYST        ; se no finisci
        Addq.w  #1,d1           ; se si sposta lo sprite
ENDJOYST:
        Rts

;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
;***************************************************************************

CancellaSchermo:
        btst    #6,2(a6)
WBlit3:
        btst    #6,2(a6)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a6)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0000,$66(a6)          ; BLTDMOD=0
        move.l  #Screen,$54(a6)         ; BLTDPT - indirizzo schermo
        move.w  #(64*256)+20,$58(a6)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo

        rts

; ***************************************************************************

                section cop,data_C

copperlist
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
```

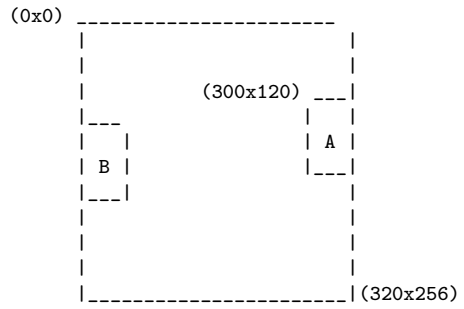
```
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w $100,$1200         ; BPLCON0 - 2 bitplanes lowres

        dc.w $180,$000  ; Color0
        dc.w $182,$aaa  ; Color1

BPLPOINTER1:
        dc.w $e0,0,$e2,0        ;primo   bitplane

        dc.l    $ffff,$fffe     ; fine della copperlist

*****************************************************************************

; il bob e' ad 1 bitplane, largo 128 pixel e alto 29 linee

Bob:
        Incbin  "Amiga.bmp"

; *****************************************************************************

; Questa e' la tabella che ci serve per "mozzare" i pixel indesiderati.

MaskRight:
        dc.w    %1111111111111111
        dc.w    %1111111111111110
        dc.w    %1111111111111100
        dc.w    %1111111111111000
        dc.w    %1111111111110000
        dc.w    %1111111111100000
        dc.w    %1111111111000000
        dc.w    %1111111110000000
        dc.w    %1111111100000000
        dc.w    %1111111000000000
        dc.w    %1111110000000000
        dc.w    %1111100000000000
        dc.w    %1111000000000000
        dc.w    %1110000000000000
        dc.w    %1100000000000000
        dc.w    %1000000000000000

; *****************************************************************************

        Section Miobuffero,BSS_C

Screen:
                ds.b    (320*256)/8

        end
```

Questo breve programma mostra come sia possibile attraverso il blitter effettuare il clipping di bob, utile in molti videogiochi. Innanzitutto vediamo cos'è il clipping in generale. Le routine di clipping sono famose sopratutto nella grafica 2d e anche 3d, infatti capita spesso di dover tracciare line che escono al di fuori della memoria video disponibile, ad esempio, si pensi ad un linea che abbia una coordinata (300,450) e che si debba disegnare in un aria video di 320x256, si nota immediatamente che la linea se disegnata con un qualunque algoritmo, quest' ultimo potrebbe

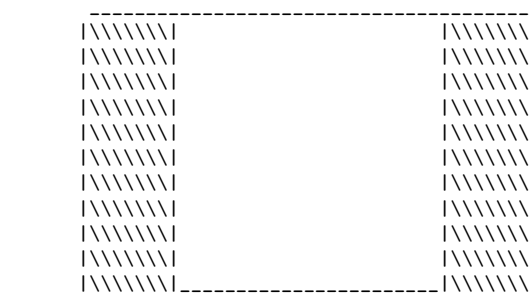
scrivere in una zona di memoria riserveta ad esempio per il codice e
quindi mandare in crash la macchina. Lo stesso discorso vale per i bob.
Infatti supponiamo di avere un' area video di dimensioni 320x256, e un bob
di 64x20 pixel, ebbene il nostro verra piazzato a video basandoce sulla
coordinate in alto a sinistra(puo' anche essere un' altra coordinata) siano
esse x e y. Attraverso il blitter potremo piazzare questo bob in un qualunque
punto dell'area disponibile, ma cosa accade se piazziamo il nostro bob
nel punto di coordinata ad esempio (300,120). Osserviamo il disegno



Come si vede la porsione del bob "B" non entra nell'area video ed esce
al di fuori. La domanda e' "Ma dove và esattamente?", la risposta e'
"dipende dai casi". Infatti supponiamo sempre di avere un' area video
di 320x256 ad 1 bitplane, ebbene finchè ci manteniamo in un range di
coordinate non c'e' nessun rischio che il blitter rovini zone di memorie
particolari, infatti la porzione di bob uscita fuori rientrera' della
sinistra, ma di un pixel piu' in basso cioe' avverà una cosa del genere.



Basti pensare al fatto che la memoria e' sequenziale, quindi arrivati
all' ultima word di una riga la succesiva word sara' la prima della riga
successiva. Quindi in questo caso si vede che c'è un rischio per i
nostri dati o il nostro codice, ma supponiamo che la cordinata sia
tremendamente vicina alla (320,256), in questo caso rischiamo davvero
grosso! In ogni modo resta un fatto, quella porzione di bob e' antiestetica.
Avete mai visto un gioco in cui i bob che escono dalla destra entrano dalla
sinitra alla Silvan ? Ci sono varie soluzion per eliminare quella porzione
di bob diventata inutile e pericolosa. Una potrebbe essere quella di fare
un area video piu' grande, cioe' aggiungere dei delle zone di sicurezza a
destra e a sinistra della memoria video. Cioe' una cosa di questo genere:



```
|\\\\| <- Area di memoria di sicurezza
|\\\\|
```

Come si vede dal disegno, questa soluzione ci garantisce due cose: la prima
e' che le porzioni di bob superflue non intaccheranno i nostri dati e la
seconda che quelle porzioni non rientreranno dalla sinistra. Ma facciamo un
po' i conti... le dimensioni di quelle aree dovranno essere al piu' uguali
alla dimesione orizontale massima dei bob, quindi se abbiamo un bob che ha
dimensione massima orizontale 128 pixel ed inoltre lo adoperiamo in un
contesto di 5 bitplane avremo bisogno di 2 aree di ((256x128)/8)*5=20480
cioè in totale 40960, inoltre dobbiamo considerare anche le aree di
sicurezza che si dovrebbero porre in alto ed in basso della nostra area
video, quindi l'occupazione di memoria sarebbe troppa.
La soluzione quindi va cercata in un algoritmo che sia in grado di prelevare
le porzioni di bob che a noi interessano in quel momento e di scriverle
nelle zone di memoria corrette. Il tutto puo' essere fatto col blitter.
Quindi il programma mostra come sia possibile realizzarlo partendo da
alcune considerazioni. Innanzitutto se il nostro bob deve essere piazzato ad
una coordinata tale che tutto il bob rientra nella memoria video allora
questo lo si puo' fare con una classica routine per spostare un bob col
blitter. Le varianti avvengono quando la coordinata xb in basso a destra
coincide con il limite massimo della memoria video e la supera totalmente.
Prepariamoci per un ragionamento abbastanza intreccioso.
Sia XM la coordinata limite della nostra area video d'ora in poi finestra,
ed inoltre supponiamo che XM sia un multiplo di 16 pixel(per comodita')
quindi facciamo delle osservazioni. Il nostro bob quando coincidera' con XM,
la coordinata xa in alto a sinistra sara' anch'essa un multiplo di 16 pixel,
poiche' il bob ha dimensione orizzontale multipla di 16 pixel. Infatti
se abbiamo un bob di 64 pixel ed XM=320 allora quando xb coincidera' con XM
si avra' che xa=320-64=256 che e' ancora un multiplo di 16, questo significa
che se il nostro bob si sposta solamente di un'altro pixel xb sara' uguale
a XM+1 ma la cosa importante e' che andremo a scrivere nel primo bit della
word successiva a (XM/16) nel nostro esempio essendo XM=320 la word che
invaderemo sara' la 41-nesima. Se avete capito questo allora avete superato
il peggio e se qualcuno di voi ha sperimentato moltissimo col blitter forse
già vede la soluzione del problema. Infatti quello che dobbiamo fare ora e'
impedire che il blitter scriva nella word invasa questo l'ho realizziamo
molto semplicemente col registro del blitter BLTLWM settendolo a
"1111111111111110", in questo modo gli ultimi bit dell'ultima word del nostro
bob non verrano copiati. Se il nostro bob si sposterà di un'altro pixel
allora la word sarà settata a "1111111111111100". Ma cosa accade se il nostro
bob esce di 16 pixel in piu' dalla finestra video ? E' evidente che non
possiamo piu' utilizzare il registro BLTLWM ma dobbiamo usufruire anche del
modulo. Se noi abbiamo un bob non in formato RAW le informazioni sono

```
memorizzate una di seguito all'altra, quindi settiamo il modulo della
sorgente a zero se ora il nostro bob e' uscito di 16 pixel fuori, allora
dobbiamo dire al nostro blitter una cosa di questo genere: il mio bob
ha oramai dimensione x-16 e altezza y quindi leggi x-16 bit e subito dopo
saltane 16(quelli fuori dalla finestra video), invece quando scrivi, scrivine
x-16 e poi salta di 320-(x-16) pixel. E' evidenete che non parliamo al
blitter in questo modo, e ne in termini di pixel ma spero di avervi fatto
capire. Quindi unendo le due tecniche della mascherazione dei bit indesiderati
e del salto delle informazioni inutili tramite il modulo riusciamo a fare
un clipping di bob velocemente, logicamente impieghiamo meno tempo a non
farlo ma pensiamo anche al fatto che in questo modo piu' porzione di bob
esce fuori e' piu' il blitter termina il lavoro di copia.

Tramite joystick potete spostare un bob di 128x29 pixel, provate a cambiare
la coordinata XMax(deve essere multiplo di 16).

In questo esempio ci limitiamo ad illustrare la tecnica del "taglio" del bob
senza preoccuparci dello sfondo. Infatti disegnamo il nostro bob mediante una
semplice copia. Inoltre per non complicare il listato, eseguiamo ogni volta
una cancellazione dell'intero schermo invece che del solo rettangolo che
racchiude il bob.
Potete provare voi ad estendere questa tecnica all'esempio del bob completo
(cioe' con ripristino dello sfondo). In questo caso dovete tenere presente
che quando il bob viene "tagliato" a destra bisogna variare il modulo e la
dimensione della blittata non solo nella routine di disegno del bob (come
avviene in questo esempio) ma anche nelle routine di salvataggio e ripristino
dello sfondo.
```

Figura 25.9: Lezione 9i5

25.10 Lezione9l1

```
; Lezione9l1.s  copia di un rettangolo tra 2 zone sovrapposte
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   copia           ; esegui la routine di copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


; ********************** LA ROUTINE DI COPIA ***************************
; viene copiato un rettangolo con larghezza=160 e altezza=20
; dalle coordinate X1=64, Y1=50 (sorgente)
; alle coordinate X2=64, Y2=40 (destinazione)
; notate che la sorgente e la destinazione si sovrappongono, e la destinazione
; si trova piu' in alto, quindi ad un indirizzo minore.
;**************************************************************************
```

```
copia:

; Carica gli indirizzi sorgente e destinazione in 2 registri

        move.l  #bitplane+((20*3*50)+64/16)*2,d0        ; ind. sorgente
        move.l  #bitplane+((20*3*40)+64/16)*2,d2        ; ind. destinazione

        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM fanno passare tutto

; carica i puntatori

        move.l  d0,$50(a5)              ; bltapt
        move.l  d2,$54(a5)              ; bltdpt

        move.l #$00140014,$64(a5)       ; bltamod e bltdmod

        move.w  #(3*20*64)+160/16,$58(a5)       ; bltsize
                                                ; altezza 20 linee e 3 planes
                                                ; largo 160 pixel (= 10 words)

        btst    #6,$02(a5)              ; aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
```

```
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.
        end

;*****************************************************************************
```

In questo esempio copiamo un rettangolo tra 2 zone sovrapposte. Poiche‘ l’indirizzo della destinazione e‘ minore di quello della sorgente (sullo schermo la destinazione si trova piu‘ in alto) possiamo tranquillamente adoperare le tecniche che abbiamo usato finora (copia di un rettangolo su schermo interleaved). Infatti le routine di questo esempio sono identiche a quelle viste nell’esempio lezione9g2.s.

Lezione9l2

```
; Lezione9l2.s  copia di un rettangolo tra 2 zone sovrapposte
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C’E‘ LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
```

```
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   copia           ; esegui la routine di copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


; ********************** LA ROUTINE DI COPIA ***************************
; viene copiato un rettangolo con larghezza=160 e altezza=20
; dalle coordinate X1=64, Y1=50 (sorgente)
; alle coordinate X2=64, Y2=55 (destinazione)
; notate che la sorgente e la destinazione si sovrappongono, e la destinazione
; si trova piu' in basso, quindi ad un indirizzo maggiore.
;**************************************************************************

copia:

; Carica gli indirizzi sorgente e destinazione in 2 registri

        move.l  #bitplane+((20*3*50)+64/16)*2,d0        ; ind. sorgente
        move.l  #bitplane+((20*3*55)+64/16)*2,d2        ; ind. destinazione

        btst    #6,2(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM fanno passare tutto

; carica i puntatori

        move.l  d0,$50(a5)              ; bltapt
        move.l  d2,$54(a5)              ; bltdpt

        move.l #$00140014,$64(a5)       ; bltamod e bltdmod

        move.w  #(3*20*64)+160/16,$58(a5)       ; bltsize
                                                ; altezza 20 linee e 3 planes
                                                ; largo 160 pixel (= 10 words)

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;**************************************************************************

        SECTION GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                ; qua carichiamo la figura in
                                ; formato RAWBLIT (o interleaved),
                                ; convertita col KEFCON.
        end

;***************************************************************************

In questo esempio copiamo un rettangolo tra 2 zone sovrapposte. Stavolta
l'indirizzo della destinazione e' maggiore di quello della sorgente (sullo
schermo la destinazione si trova piu' in basso).
Come potete vedere il risultato ottenuto non e' quello che desideravamo.
```

Lezione9l3

```
; Lezione9l3.s  copia di un rettangolo tra 2 zone sovrapposte usando il modo
;               DISCENDENTE.
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************
```

```
                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   copia           ; esegui la routine di copia

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


; ********************** LA ROUTINE DI COPIA ***************************
; viene copiato un rettangolo con larghezza=160 e altezza=20
; dalle coordinate X1=64, Y1=50 (sorgente)
; alle coordinate X2=64, Y2=55 (destinazione)
; la sorgente e la destinazione si sovrappongono e la destinazione ha un
; indirizzo maggiore (piu' in basso sullo schermo).
; Per effettuare correttamente la copia si usa il MODO DISCENDENTE
;**************************************************************************

copia:

; Carica gli indirizzi sorgente e destinazione in 2 registri
; NOTATE LA DIFFERENZA RISPETTO AL CASO NORMALE:
; gli indirizzi sono quelli delle word piu' in basso a destra dei rettangoli.
; Se Xa e Ya sono le coordinate dell'angolo superiore sinistro, la coordinata
; Yb della riga a cui appartiene la riga piu' in basso del rettangolo e':
; Yb=Ya+ALTEZZA RETTANGOLO
; pertanto, nel calcolo dell'indirizzo, l'OFFSET relativo alla Y e' dato da:
; OFFSET_Y = (Yb*(NUMERO WORDS PER RIGA)*(NUMERO PLANES))*2.
; L'offset relativo alla X invece si calcola osservando che
```

```
; Xa+LARGHEZZA RETTANGOLO e‘ la coordinata X del primo pixel della word che
; si trova FUORI del rettangolo immediatamente a destra. L’OFFSET di tale word
; e‘ quindi ((Xa+LARGHEZZA RETTANGOLO)/16)*2. A noi pero‘ non interessa questa
; word ma invece quella che la precede, ovvero l’ultima word a destra del
; rettangolo, il cui OFFSET e‘ quindi:
; OFFSET_Y = ((Xa+LARGHEZZA RETTANGOLO)/16-1)*2

        move.l  #bitplane+((20*3*(20+50))+(160+64)/16-1)*2,d0   ; ind. sorgente
        move.l  #bitplane+((20*3*(20+55))+(160+64)/16-1)*2,d2   ; ind. dest.

        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$09f00002,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; MODO DISCENDENTE!!!!!

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo

; carica i puntatori

        move.l  d0,$50(a5)              ; bltapt
        move.l  d2,$54(a5)              ; bltdpt

; Questa istruzione setta i moduli della sorgente e della destinazione.
; Come abbiamo spiegato NON CI SONO DIFFERENZE RISPETTO AL CASO ASCENDENTE!

        move.l #$00140014,$64(a5)       ; bltamod e bltdmod

; anche per quanto riguarda la dimensione non ci sono differenze

        move.w  #(3*20*64)+160/16,$58(a5)       ; bltsize
                                                ; altezza 20 linee e 3 planes
                                                ; largo 160 pixel (= 10 words)

        btst    #6,$02(a5)      ; aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C’E‘ UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
```

```
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                        ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.
        end

;***************************************************************************
```

In questo esempio copiamo un rettangolo tra 2 zone sovrapposte. L'indirizzo della destinazione e' maggiore di quello della sorgente (sullo schermo la destinazione si trova piu' in basso) e pertanto usiamo il modo discendente. Il modo discendente viene attivato settando a 1 il bit 1 del registro BLTCON1. L'unica differenza rispetto al caso ascendente e' nel calcolo degli indirizzi da scrivere nei puntatori ai canali DMA, per il quale si applicano le formule spiegate nella lezione.

25.11 Lezione9m1

```
; Lezione9m1.s  Esempio di uso delle Mask con il modo discendente
; premete alternativamente i tasti destro e sinistro del mouse per vedere
; varie blittate con maschere diverse.


        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
```

```
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; prepara i parametri

        move.w  #$ffff,d0               ; maschera prima word (piu' a destra)
                                        ; fa passare tutti i bit
        move.w  #$ffff,d1               ; maschera ultima word (piu' a sinistra
                                        ; fa passare tutti i bit
        move.l  #bitplane+7*40+6,a0     ; indirizzo destinazione
        bsr.w   Copia

mouse2:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse2

; prepara i parametri

        moveq   #$0000,d0               ; maschera prima word
                                        ; cancella tutto
        move.w  #$ffff,d1               ; maschera ultima word
                                        ; fa passare tutti i bit
        move.l  #bitplane+37*40+6,a0    ; indirizzo destinazione
        bsr.s   Copia
mouse3:
        btst    #6,$bfe001              ; mouse premuto?
        bne.s   mouse3

; prepara i parametri

        move.w  #%1010101010101010,d0   ; maschera prima word
                                        ; un bit si e uno no
        move.w  #%0000000000000001,d1   ; maschera ultima word
                                        ; solo bit piu' a destra
        move.l  #bitplane+67*40+6,a0    ; indirizzo destinazione
        bsr.s   Copia

mouse4:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse4

; prepara i parametri

        move.w  #$0000,d0               ; maschera prima word
                                        ; cancella tutto
        move.w  #$0000,d1               ; maschera ultima word
                                        ; cancella tutto
        move.l  #bitplane+97*40+6,a0    ; indirizzo destinazione
        bsr.s   Copia

mouse5:
        btst    #6,$bfe001              ; mouse premuto?
        bne.s   mouse5

; prepara i parametri
```

```
        move.w  #%1111000011110000,d0   ; maschera prima word
                                        ; 4 bit si e 4 no
        move.w  #%0000011010011100,d1   ; maschera ultima word
                                        ; fa passare solo i bit 2,3,4,7,9 e 10
        move.l  #bitplane+127*40+6,a0   ; indirizzo destinazione
        bsr.s   Copia

mouse6:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   mouse6

; prepara i parametri

        move.w  #%0000000001111111,d0   ; maschera prima word
                                        ; cancella i 9 bit piu‘ a sinistra
        move.w  #%1111111000000000,d1   ; maschera ultima word
                                        ; cancella i 9 bit piu‘ a destra
        move.l  #bitplane+157*40+6,a0   ; indirizzo destinazione
        bsr.s   Copia

mouse:
        btst    #6,$bfe001
        bne.s   mouse

        rts

;*****************************************************************************
; Questa routine copia la figura sullo schermo in modo discendente
; Prende come parametri
; A0 - indirizzo destinazione
; D0 - maschera prima word
; D1 - maschera ultima word
;*****************************************************************************

Copia:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  d0,$44(a5)              ; BLTAFWM carica il parametro
        move.w  d1,$46(a5)              ; BLTALWM carica il parametro
        move.w  #$09f0,$40(a5)          ; BLTCON0 (usa A+D)
        move.w  #$0002,$42(a5)          ; BLTCON1 modo discendente
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  #figura+7*6-2,$50(a5)   ; BLTAPT  (fisso alla figura sorgente)
                                        ; puntiamo l’ultima word della figura
                                        ; per via del modo discendente
        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*7)+3,$58(a5)       ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
        rts                             ; altezza 7 linee (1 plane)

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
```

```
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $100,$1200      ; bplcon0 - 1 bitplane Lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Definiamo in binario la figura, che e' larga 3 words, e alta 7 linee

Figura:
;                0123456789012345  0123456789012345  0123456789012345
        dc.w    %1111111111000000,%0000001111000000,%0000001111111111
        dc.w    %1111111111000000,%0000111111110000,%0000001111111111
        dc.w    %1111111111000000,%0011111111111100,%0000001111111111
        dc.w    %1111111111111111,%1111111111111111,%1111111111111111
        dc.w    %1111111111000000,%0011111111111100,%0000001111111111
        dc.w    %1111111111000000,%0000111111110000,%0000001111111111
        dc.w    %1111111111000000,%0000001111000000,%0000001111111111

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

Questo esempio e' quasi identico all' esempio lezione9h1.s. L'unica differenza
e' che la copia viene effettuata in modo discendente. Blittando in questo modo
la prima word di ogni riga e' la riga che appare piu' a destra sullo schermo,
e l'ultima e' la word che appare piu' a sinistra. Pertanto, al contrario di
quanto accade usando il modo ascendente, la maschera della prima word
(contenuta in BLTAFWM) si applica alla word piu' a destra e la maschera
dell'ultima word (BLTALWM) alla word piu' a sinistra.
Eseguendo il programma vedrete come le maschere si applichino in modo
rovesciato rispetto all' esempio lezione9h1.s.
```

Lezione9m2

```
; Lezione9m2.s  Sparizione tramite scorrimento verso sinistra di un immagine
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        bsr.s   Scorri          ; esegui la routine di scorrimento

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


;*****************************************************************************
; Questa routine fa scomparire progressivamente un immagine
; facendola scorrere verso sinistra
;*****************************************************************************

Scorri:
        move.w  #160-1,d7       ; il loop va eseguito una volta per ogni pixel
                                ; l'immagine e' larga 160 pixel (10 words)

; In questo esempio copiamo un'immagine su se stessa, ma shiftandola
; continuamente di un pixel, in modo da farla scorrere.
; Pertanto gli indirizzi sorgente e destinazione sono uguali.
; Per shiftare verso sinistra usiamo il modo discendente, e quindi
; puntiamo l'ultima word dell'immagine:

        move.l  #bitplane+((20*3*(50+20))+(64+160)/16-1)*2,d0   ; ind. sorg. e
                                                                ; destinazione
```

```
ScorriLoop:

; Aspetta il vblank in modo da far scorrere l'immagine di un pixel ad
; ogni fotogramma.

WaitWblank:
        CMP.b   #$ff,$dff006            ; aspetta la linea 255
        bne.s   WaitWblank
Aspetta:
        CMP.b   #$ff,$dff006            ; ancora linea 255 ?
        beq.s   Aspetta

        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit:
        btst    #6,2(a5)
        bne.s   waitblit

        move.l  #$19f00002,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; in modo discendente con shift
                                        ; (verso sinistra) di un pixel

        move.l  #$ffff7fff,$44(a5)      ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $7fff = %0111111111111111
                                        ;   cancella il bit piu' a sinistra
; carica i puntatori

        move.l  d0,$50(a5)                      ; bltapt - sorgente
        move.l  d0,$54(a5)                      ; bltdpt - destinazione

; il modulo e' calcolato come al solito

        move.l #$00140014,$64(a5)               ; bltamod e bltdmod
        move.w  #(3*20*64)+160/16,$58(a5)       ; bltsize
                                                ; altezza 20 linee e 3 planes
                                                ; largo 160 pixel (= 10 words)

        dbra    d7,ScorriLoop                   ; ripeti per ogni pixel

        btst    #6,$02(a5)              ; aspetta che il blitter finisca
waitblit2:
        btst    #6,$02(a5)
        bne.s   waitblit2
        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C'E' UNA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.
```

```
        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


BITPLANE:
        incbin  "assembler2:sorgenti6/amiga.rawblit"
                                ; qua carichiamo la figura in
                                ; formato RAWBLIT (o interleaved),
                                ; convertita col KEFCON.
        end

;***************************************************************************
```

In questo esempio riproponiamo l'effetto visto in lezione9h4.s solo che effettuiamo lo scorrimento verso sinistra. Per shiftare verso sinistra, dobbiamo usare il blitter in modo discendente. La blittata e' sempre una copia di un immagine su se stessa (indirizzi sorgente e destinazione uguali) ma lo shift avviene verso sinistra. Dobbiamo mascherare la word piu' a sinistra per cancellare la colonna di pixel piu' a sinistra. In modo discendente la word piu' a sinistra e' mascherata da BLTALWM. Per cancellare la colonna di pixel piu' a sinistra dobbiamo porre in BLTALWM il valore %0111111111111111 che cancella il bit piu' a sinistra e lascia passare gli altri.
Notate la differenza con lezione9h4 dove invece mascheravamo il bit piu' a destra.

25.12 Lezione9n1

```
; Lezione9n1.s  Signore e signori, lo SCROLLTEXT!!!!!
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
```

```
        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP

        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     testo(pc),a0            ; punta al testo dello scrolltext

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        bsr.s   printchar               ; routine che stampa i nuovi chars
        bsr.s   Scorri                  ; esegui la routine di scorrimento

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse2:
        rts

;*****************************************************************************
; Questa routine stampa un carattere. Il carattere viene stampato in una
; parte di schermo invisibile.
; A0 punta al testo da stampare.
;*****************************************************************************

PRINTCHAR:
        subq.w  #1,contatore    ; diminuisci il contatore di 1
        bne.s   NoPrint         ; se e' diverso da 0, non stampiamo,
        move.w  #16,contatore   ; altrimenti si; reinizializza il contatore

        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        bne.s   noreset         ; Se e' diverso da 0 stampalo,
        lea     testo(pc),a0    ; altrimenti ricomincia il testo daccapo
        MOVE.B  (A0)+,D2        ; Primo carattere in d2
noreset
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        ADD.L   D2,D2           ; MOLTIPLICA PER 2 IL NUMERO PRECEDENTE,
                                ; perche' ogni carattere e' largo 16 pixel
```

```
        MOVE.L  D2,A2

        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        btst    #6,$02(a5)      ; aspetta che il blitter finisca
waitblit:
        btst    #6,$02(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0: copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo

        move.l  a2,$50(a5)                      ; BLTAPT: indirizzo font
        move.l  #bitplane+50*42+40,$54(a5)      ; BLTDPT: indirizzo bitplane
                                                ; fisso, fuori dalla parte
                                                ; visibile dello schermo.
        move    #120-2,$64(a5)                  ; BLTAMOD: modulo font
        move    #42-2,$66(a5)                   ; BLTDMOD: modulo bit planes
        move    #(20<<6)+1,$58(a5)              ; BLTSIZE: font 16*20
NoPrint:
        rts

contatore
        dc.w    16

;***************************************************************************
; Questa routine fa scorrere il testo verso sinistra
;***************************************************************************

Scorri:

; Gli indirizzi sorgente e destinazione sono uguali.
; Shiftiamo verso sinistra, quindi usiamo il modo discendente.

        move.l  #bitplane+((21*(50+20))-1)*2,d0         ; ind. sorgente e
                                                        ; destinazione
ScorriLoop:
        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit2:
        btst    #6,2(a5)
        bne.s   waitblit2

        move.l  #$19f00002,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l  #$ffff7fff,$44(a5)      ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $7fff = %0111111111111111
                                        ;   cancella il bit piu‘ a sinistra

; carica i puntatori

        move.l  d0,$50(a5)                      ; bltapt - sorgente
        move.l  d0,$54(a5)                      ; bltdpt - destinazione

; facciamo scorrere un immagine larga tutto lo schermo, quindi
; il modulo e‘ azzerato.

        move.l  #$00000000,$64(a5)              ; bltamod e bltdmod
        move.w  #(20*64)+21,$58(a5)             ; bltsize
                                                ; altezza 20 linee, largo 21
        rts                                     ; words (tutto lo schermo)
```

```
; Questo e' il testo. con lo 0 si termina. Il font usato ha solo i caratteri
; maiuscoli, attenzione!

testo:
        dc.b    "ECCO FINALMENTE LO SCROLLTEXT, CHE TUTTI STAVANO"
        dc.b    " ASPETTANDO... IL FONT E' DI 16*20 PIXEL!..."
        dc.b    " LO SCROLL AVVIENE CON TRANQUILLITA'...",0


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

        dc.w    $108,2          ; Il bitplane e' largo 42 bytes, ma solo 40
                                ; bytes sono visibili, quindi il modulo
                                ; vale 42-40=2
;       dc.w    $10a,2          ; Usiamo un solo bitplane, quindi BPLMOD2
                                ; non e' necessario

        dc.w    $100,$1200      ; bplcon0 - 1 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0

; queste istruzioni della copperlist cambiano il colore 1 ogni 2 righe

        dc.w    $5e01,$fffe     ; prima riga scrolltext
        dc.w    $0182,$f50      ; color1
        dc.w    $6001,$fffe
        dc.w    $0182,$d90
        dc.w    $6201,$fffe
        dc.w    $0182,$dd0
        dc.w    $6401,$fffe
        dc.w    $0182,$5d2
        dc.w    $6601,$fffe
        dc.w    $0182,$2f4
        dc.w    $6801,$fffe
        dc.w    $0182,$0d7
        dc.w    $6a01,$fffe
        dc.w    $0182,$0dd
        dc.w    $6c01,$fffe
        dc.w    $0182,$07d
        dc.w    $6e01,$fffe
        dc.w    $0182,$22f
        dc.w    $7001,$fffe
        dc.w    $0182,$40d
        dc.w    $7201,$fffe
        dc.w    $0182,$80d

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
;***************************************************************************

; Qui e' memorizzato il FONT di caratteri 16x20

FONT:
        incbin  "assembler2:sorgenti6/font16x20.raw"

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    42*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio presentiamo uno degli effetti piu' classici delle demo:
lo scrolltext. Si tratta di un testo che scorre sullo schermo da destra
verso sinistra, e che di solito contiene i saluti (greeetings) mandati
dagli autori della demo agli altri demo-coders. Come si realizza uno
scrolltext? Si potrebbe pensare di stampare tutto il testo su un bitplane
piu' grande dello schermo visibile e poi scrollare il bitplane. Questa
tecnica ha lo svantaggio di occupare molta memoria, perche' ha bisogno
di un bitplane che contenga tutto il testo.
Noi usiamo un'altra tecnica basata sul blitter, per la quale basta disporre
di un bitplane largo appena 16 pixel (1 word) in piu' dell'area visibile.
Abbiamo dunque una colonna di una word invisibile alla destra dello schermo.
Supponiamo di stampare un carattere nella parte invisibile del bitplane, e
contemporaneamente di far scorrere verso sinistra il bitplane mediante il
blitter. Accade, come potete immaginare, che il carattere si sposti un pixel
alla volta verso sinistra.
Notate che i puntatori ai bitplane rimangono sempre fissi.
Quando il carattere e' completamente visiblie, il che accade dopo 16
shiftate di 1 pixel, essendo il carattere largo 16 pixel, possiamo stampare
il carattere successivo nella parte invisibile dello schermo.
I caratteri che arrivano al bordo sinistro vengono cancellati dalla maschera
del blitter.
In pratica l'effetto e' ottenuto usando 2 routine.
La prima "Printchar" si occupa di stampare i caratteri sullo schermo.
La stampa deve avvenire solo quando il carattere precedentemente stampato e'
diventato completamente visibile, in modo da evitare di sovrapporre i 2
caratteri.
Siccome ogni carattere e' largo 16 pixel, e il testo scorre un pixel alla
volta, in sostanza la stampa deve avvenire ogni 16 volte che la routine viene
chiamata.
Per questo viene usato un contatore che viene decrementato ad ogni chiamata.
Quando esso assume il valore 0 il carattere viene stampato, e il contatore
reinizializzato a 16. La stampa vera e propria e' una semplice copia con il
blitter (usato in modo ascendente) fatta nella maniera che conosciamo.
La routine "Scorri" e' incaricata di far scorrere il testo usando lo shift
del blitter verso sinistra (cioe' in modo discendente) nel modo che abbiamo
visto nell' esempio lezione9m2.s. Poiche' i caratteri sono alti 20 righe,
tutto il testo occupa una "fascia" di schermo alta 20 righe.
E' necessario far scorrere tutta questa "fascia", ovvero un rettangolo alto 20
righe e largo quanto tutto il bitplane (compresa naturalmente la parte
INVISIBILE, in modo da far scorrere i caratteri nella parte visibile).
La maschera dell'ultima word provvede a cancellare i caratteri che raggiungono
il bordo sinistro.
Abbiamo usato uno schermo formato da 1 bitplane, e i colori sono ottenuti
tramite il copper (altrimenti che furbi saremmo?).

Lezione9n2

```
; Lezione9n2.s  Ancora Scrolltext!! Quello nell'intro del disco 1!
;               Tasto sinistro per uscire.

        Section BigScroll,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; blitter, copper, bitplane DMA

START:

;        PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L  #BITPLANE+100*44,d0     ; bitplane
        LEA     BPLPOINTERS,A1
        MOVEQ   #3-1,D1                 ; numero di bitplanes
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #44*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTB

        bsr.s   makecolors              ; Fai effetto in copperlist

        lea     $dff000,a6
        MOVE.W  #DMASET,$96(a6)         ; DMACON - abilita dma
        move.l  #COPPERLIST,$80(a6)     ; Puntiamo la nostra COP
        move.w  d0,$88(a6)              ; Facciamo partire la COP
        move.w  #0,$1fc(a6)             ; Disattiva l'AGA
        move.w  #$c00,$106(a6)          ; Disattiva l'AGA
        move.w  #$11,$10c(a6)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   MainScroll      ; routine di gestione scroll

        BTST.B  #6,$bfe001      ; tasto del mouse premuto?
        BNE.s   mouse
        rts

;*****************************************************************************
```

```
; questa routine crea una sfumatura di colore nella copperlist
; in pratica nella copperlist c'e' dello spazio vuoto che viene
; riempito da questa routine che mette le giuste istruzioni copper.
; Vedremo molte di queste routines in Lezione11.txt
;***************************************************************************

MAKECOLORS:
        lea     scol,a0         ; Indirizzo dove modificare copperlist
        lea     coltab,a1       ; tabella colori 1
        lea     coltab2,a2      ; tabella colori 2
        move.l  #$a807,d1       ; riga di partenza = $A0
        moveq   #63,d0          ; numero di righe
col1:
        move.w  d1,(a0)+        ; crea istruzione WAIT
        move.w  #$fffe,(a0)+

        move.w  #$0182,(a0)+    ; istruzione che modifica COLOR01
        move.w  (a1)+,(a0)+
        move.w  #$018E,(a0)+    ; istruzione che modifica COLOR07
        move.w  (a2)+,(a0)+

        add.w   #$100,d1        ; prossima riga
        dbra    d0,col1
        rts

coltab:
         dc.w   $00,$11,$22,$33,$44,$55,$66,$77,$88,$99
         dc.w   $aa,$bb,$cc,$dd,$ee,$ff,$ff,$ee,$dd,$cc
         dc.w   $bb,$aa,$99,$88,$77,$66,$55,$44,$33,$22
         dc.w   $11,$00
         dc.w   $000,$110,$220,$330,$440,$550,$660,$770,$880,$990
         dc.w   $aa0,$bb0,$cc0,$dd0,$ee0,$ff0,$ff0,$ee0,$dd0,$cc0
         dc.w   $bb0,$aa0,$990,$880,$770,$660,$550,$440,$330,$220
         dc.w   $110,$000
         dc.w   $000,$101,$202,$303,$404,$505,$606,$707,$808,$909
         dc.w   $a0a,$b0b,$c0c,$d0d,$e0e,$f0f,$f0f,$e0e,$d0d,$c0c
         dc.w   $b0b,$a0a,$909,$808,$707,$606,$505,$404,$303,$202
         dc.w   $101,$000,0,0

coltab2:
         dc.w   $000,$101,$202,$303,$404,$505,$606,$707,$808,$909
         dc.w   $a0a,$b0b,$c0c,$d0d,$e0e,$f0f,$f0f,$e0e,$d0d,$c0c
         dc.w   $b0b,$a0a,$909,$808,$707,$606,$505,$404,$303,$202
         dc.w   $101,$000,0,0
         dc.w   $000,$011,$022,$033,$044,$055,$066,$077,$088,$099
         dc.w   $0aa,$0bb,$0cc,$0dd,$0ee,$0ff,$0ff,$0ee,$0dd,$0cc
         dc.w   $0bb,$0aa,$099,$088,$077,$066,$055,$044,$033,$022
         dc.w   $011,$000
         dc.w   $000,$110,$220,$330,$440,$550,$660,$770,$880,$990
         dc.w   $aa0,$bb0,$cc0,$dd0,$ee0,$ff0,$ff0,$ee0,$dd0,$cc0
         dc.w   $bb0,$aa0,$990,$880,$770,$660,$550,$440,$330,$220
         dc.w   $110,$000

;***************************************************************************
;               ROUTINE PRINCIPALE DELLO SCROLLTEXT
;***************************************************************************

MainScroll:
        lea     $dff000,a6
        btst.b  #10,$16(a6)             ; premuto il tasto destro?
        beq.s   SaltaScroll             ; se si, sposta in verticale la scritta
                                        ; senza scrollarla
```

```
        move.l  noscroll(pc),d0         ; contatore di scroll
        subq.l  #1,d0                   ; decrementa il contatore
        bmi.s   do_scrolling            ; se negativo scrolla
        move.l  d0,noscroll             ; altrimenti sposta solo la scritta
        bra.s   SaltaScroll

do_scrolling:                           ; effettua lo scrolling
        clr.l   noscroll                ; azzera il contatore

        bsr.w   PrintChar               ; stampa nuovo carattere
        bsr.s   DoScroll                ; scrolla text

SaltaScroll:
        bsr.s   Drawscroll              ; richiama la routine che disegna
                                        ; il testo sullo schermo

        lea     sinustab(PC),a0         ; queste istruzioni ruotano i valori
        lea     4(a0),a1                ; della tabella delle posizioni
        move.l  (a0),d0                 ; verticali dello scrolltext
copysinustab:
        move.l  (a1)+,(a0)+
        cmpi.l  #$ffff,(a1)             ; Flag di fine tabella? Se non ancora,
        bne.s   copysinustab            ; continua a spostare...
        move.l  d0,(a0)                 ; Alla fine, metti il primo val. in
        rts                             ; fondo!

;***************************************************************************
; Questa routine effettua lo scrolling vero e proprio. Da notare che per
; stabilire la velocita' di scroll, si usa la label "speedlogic", che non
; e' altro che il valore da inserire nel BLTCON0, che a seconda delle varie
; velocita' ha un valore di shift diverso.
;***************************************************************************

DoScroll:
        lea     BITPLANE+2,a0   ; Source (16 pixel piu' avanti)
        lea     BITPLANE,a1     ; Dest   (inizio... quindi <- di 16 pixel!)
        moveq   #3-1,d7         ; Numero blittate = 3 per 3 planes
BlittaLoop1:
        btst    #6,2(a6)        ; dmaconr - waitblit
bltx:
        btst    #6,2(a6)        ; dmaconr - waitblit
        bne.s   bltx

        moveq   #0,d1
        move.w  d1,$42(a6)              ; BLTCON1
        move.l  d1,$64(a6)              ; BLTAMOD, BLTDMOD
        moveq   #-1,d1                  ; $FFFFFFFF
        move.l  d1,$44(a6)              ; BLTAFWM, BLTALWM
        move.w  speedlogic(PC),$40(a6)  ; BLTCON0 (stabilisce velocita' di
                                        ;          scroll tramite lo shift)

        btst    #6,2(a6)        ; dmaconr - waitblit
blt23:
        btst    #6,2(a6)        ; dmaconr - waitblit
        bne.s   blt23

        move.l  a0,$50(a6)              ; BLTAPT
        move.l  a1,$54(a6)              ; BLTDPT
        move.w  #(32*64)+22,$58(a6)     ; BLTSIZE

        add.w   #32*44,a0       ; prossimo plane sorgente
```

```
        add.w   #32*44,a1       ; prossimo plane destinazione

        dbra    d7,BlittaLoop1
        rts

;***************************************************************************
; questa routine disegna sullo schermo lo scrolltext ad una posizione verticale
; variabile secondo i valori di una sinusoide (ossia una bella SIN TAB!).
; Da notare che anziche' copiarlo con lente blittate, avremmo potuto in modo
; piu' "economico" e "amighevole" cambiare solo i puntatori ai bitplanes,
; facendo lo stesso lavoro con pochi move. Pero' questo e' un sorgente
; dedicato al blitter, quindi blittiamo!
;***************************************************************************

Drawscroll:
        lea     BITPLANE,a0             ; puntatore sorgente
        lea     sinustab(pc),a5         ; tabella seno
        move.l  (a5),d4                 ; leggi coordinata Y
                                        ; (la prima della tabella)
        lea     BITPLANE+(112*44),a5    ; indirizzo destinazione
        add.l   d4,a5                   ; aggiungi coordinata Y

        btst    #6,2(a6)                ; aspetta che il blitter sia fermo
blt1e:                                  ; prima di modificare i registri
        btst    #6,2(a6)
        bne.s   blt1e

        moveq   #-1,d1
        move.l  d1,$44(a6)              ; BLATLWM, BLTAFWM
        moveq   #0,d1
        move.l  d1,$64(a6)              ; BLTAMOD/BLTDMOD
        move.l  #$09f00000,$40(a6)      ; BLTCON0 - copia normale

        moveq   #3-1,d7                 ; Num. di planes
copialoopa:
        btst    #6,2(a6)                ; aspetta che il blitter sia fermo
blt1f:
        btst    #6,2(a6)
        bne.s   blt1f

        move.l  a0,$50(a6)              ; BLTAPT
        move.l  a5,$54(a6)              ; BLTDPT
        move.w  #32*64+22,$58(a6)       ; BLTSIZE - copyscroll

        add.w   #32*44,a0               ; prossimo plane sorgente
        add.w   #256*44,a5              ; prossimo plane dest.

        dbra    d7,copialoopa
        rts

; Questa tabella contiene gli offset per le coordinte Y per far andare su e
; giu' lo scroll.

sinustab:
        dc.l    0,44,88,132,176,220,264,308,352,396
        dc.l    440,484,528,572,616,660,704,748,792
        dc.l    836,880,924,968,1012,1056,1100,1144,1188,1232
        dc.l    1276,1276,1232
        dc.l    1188,1144,1100,1056,1012,968,924,880,836,792,748,704
        dc.l    660,616,572,528,484,440,396,352,308
        dc.l    264,220,176,132,88,44,0
sinusend:
```

```
        dc.l   0
        dc.l   $ffff   ; flag di fine tabella


;***************************************************************************
; Questa routine stampa i nuovi caratteri. Da notare che nel testo ci sono
; anche dei FLAG, in questo caso dei numeri da 1 a 5, che cambiano la
; velocita' di scroll. Questo cambiando il valore di shift da mettere in
; bltcon0, nonche' il numero di caratteri da stampare ogni frame (e' chiaro
; che a velocita' supersonica occorre stampare piu' caratteri al frame!).
; Altro particolare da notare, e' che il sistema usato per costruire il
; font e' diverso da quelli visti fino ad ora. Infatti il font caratteri
; non e' altro che una schermata 320*200 a 8 colori, con i caratteri posti
; l'uno accanto all'altro, e una fila sotto l'altra. Questo rende piu' facile
; disegnarsi un proprio font, ma richiede una routine diversa per trovare il
; font. Infatti, occorre fare una tabella contenente gli offsets dall'inizio
; del font di ogni carattere, e a seconda del valore ascii che dobbiamo
; stampare, prendere il corrispondente valore dalla tabella per sapere
; la posizione del carattare in questione. Cio' puo' sembrare complesso, ma
; dato che i caratteri nel font sono messi nell'ordine ascii, vedrete che e'
; molto semplice scriversi la tabella con gli offset!
; Il font, comunque, e' presente anche in formato .iff, in modo da rendere
; piu' chiuaro il sistema, e piu' semplice il disegno di un nuovo font.
;***************************************************************************

PrintChar:
        tst.w   textctr         ; se il contatore e‘ positivo non stampa
        bne.w   noPrint
        move.l  textptr(PC),a0  ; legge il carattere da stampare
        moveq   #0,d0
        move.b  (a0)+,d0
        cmp.l   #textend,textptr        ; siamo alla fine del testo ?
        bne.s   noend
        lea     scrollmsg(PC),a0        ; se si ricomincia da capo!
        move.b  (a0)+,d0                ; carattere in d0
noend:
        cmp.b   #1,d0                   ; FLAG 1? Allora speed = 1
        bne.s   nots1
        move.w  #32,scspeed             ; valore iniziale di textctr
        move.w  #$f9f0,speedlogic       ; valore di BPLCON0
        move.b  (a0)+,d0                ; prossimo carattere in d0
        bra.s   contscroll
nots1:
        cmpi.b  #2,d0                   ; FLAG 2? Allora speed = 2
        bne.s   nots2
        move.w  #16,scspeed
        move.w  #$e9f0,speedlogic       ; valore di BPLCON0
        move.b  (a0)+,d0
        bra.s   contscroll
nots2:
        cmpi.b  #3,d0                   ; FLAG 3? Allora speed = 3
        bne.s   nots3
        move.w  #8,scspeed
        move.w  #$c9f0,speedlogic       ; valore di BPLCON0
        move.b  (a0)+,d0
        bra.s   contscroll
nots3:
        cmpi.b  #4,d0                   ; FLAG 4? Allora speed = 4
        bne.s   nots4
        move.w  #4,scspeed
        move.w  #$89f0,speedlogic       ; valore di BPLCON0
        move.b  (a0)+,d0
```

```
        bra.s   contscroll
nots4:
        cmpi.b  #5,d0                   ; Flag 5? Allora speed = 5
        bne.s   contscroll
        move.w  #2,scspeed
        move.w  #$19f0,speedlogic       ; valore di BPLCON0
        move.b  (a0)+,d0

; Qua, passato il controllo dei flag, si stampa il carattere. Si noti il modo
; in cui si trova il carattere, tramite la tabella con gli offsets.

contscroll:
        move.l  a0,textptr      ; salva il puntatore al prossimo carattere
        subi.b  #$20,d0         ; ascii - 20 = il primo carattere e' lo spazio
        lsl.w   #2,d0           ; moltiplica * 4 per trovare l'address in tab,
                                ; dato che ogni valore in tab e' .L (4 byte)
        lea     addresses(PC),a0
        move.l  0(a0,d0.w),a0   ; copia in a0 l'indirizzo del carattere, preso
                                ; dalla tabella.

        btst    #6,2(a6)        ; dmaconr - waitblit
blt30:
        btst    #6,2(a6)        ; dmaconr - waitblit
        bne.s   blt30

        moveq   #-1,d1
        move.l  d1,$44(a6)              ; BLTALWM, BLTAFWM
        move.l  #$09F00000,$40(a6)      ; BLTCON0/1 - copia normale
        move.l  #$00240028,$64(a6)      ; BLTAMOD = 36, BLTDMOD = 40

        lea     BITPLANE+40,a1          ; puntatore destinazione
        moveq   #3-1,d7                 ; num. bitplanes
CopyCharL:

        btst    #6,2(a6)        ; dmaconr - waitblit
blt31:
        btst    #6,2(a6)        ; dmaconr - waitblit
        bne.s   blt31

        move.l  a0,$50(a6)              ; BLTAPT (carattere in font)
        move.l  a1,$54(a6)              ; BLTDPT (bitplane)
        move.w  #32*64+2,$58(a6)        ; BLTSIZE

        add.w   #32*44,a1       ; prossimo bitplane destinazione
        lea     40*200(a0),a0   ; 1 bitplane della pic che contiene il font

        dbra    d7,copycharL

        move.w  scspeed(PC),textctr     ; valore iniziale contatore stampa
noPrint:
        subq.w  #1,textctr      ; decrementa il contatore che indica quando
                                ; stampare
endPrint:
        rts

; variabili

textptr:         dc.l   scrollmsg       ; puntatore al carattere da stampare

textctr:         dc.w   0               ; contatore che indica quando stampare
noscroll:        dc.l   0               ; contatore che indica quando scrollare
```

```
scspeed:        dc.w   0              ; valore iniziale del contatore che
                                      ; indica quando stampare
                                      ; varia a seconda della velocita'

speedlogic:     dc.w   0              ; valore di BLTCON0
                                      ; varia a seconda della velocita'
                                      ; perche' varia il valore di shift

;***************************************************************************
; Questa tabella contiene una serie di indirizzi del font, che indicano la
; posizione dei caratteri ascii nel font stesso: per esempio, il primo e'
; Bigf senza altri offset, infatti il primo carattere che si trova nel font
; e' lo spazio, che e' anche il primo dell'ascii. Il secondo (che in ascii e'
; il punto esclamativo !) e' a bigf+4, infatti il ! nel font si trova al
; secondo posto, ossia 4 bytes (32 pixel) dopo il primo, essendo ogni
; carattere largo (e alto) 32 pixel.
; Dato che il font si trova in una figura 320*200, ci potranno stare solo
; 10 caratteri per fila orizzontale, per cui i caratteri dall'11 al 20
; dovranno essere in una fila sotto, quelli dal 22 al 30 sotto ancora, e
; cosi' via.
;***************************************************************************

addresses:
        dc.l BigF      ; primo carattere: " "
        dc.l BigF+4    ; secondo carattere: "!"
        dc.l BigF+8
        dc.l BigF+12,BigF+16,BigF+20,BigF+24,BigF+28,BigF+32,BigF+36

; seconda fila di caratteri nel font: si parte da 1280, ossia 32*40, infatti
; occorre saltare le 32 linee di altezza della prima fila di caratteri

        dc.l BigF+1280         ; undicesimo carattere: "
        dc.l BigF+1284
        dc.l BigF+1288
        dc.l BigF+1292
        dc.l BigF+1296,BigF+1300,BigF+1304,BigF+1308,BigF+1312,BigF+1316

; terza fila di caratteri nel font

        dc.l BigF+2560,BigF+2564,BigF+2568,BigF+2572,BigF+2576,BigF+2580
        dc.l BigF+2584,BigF+2588,BigF+2592,BigF+2596
; quarta
        dc.l BigF+3840,BigF+3844,BigF+3848,BigF+3852,BigF+3856,BigF+3860
        dc.l BigF+3864,BigF+3868,BigF+3872,BigF+3876
; quinta
        dc.l BigF+5120,BigF+5124,BigF+5128,BigF+5132,BigF+5136,BigF+5140
        dc.l BigF+5144,BigF+5148,BigF+5152,BigF+5156
; sesta
        dc.l BigF+6400,BigF+6404,BigF+6408,BigF+6412,BigF+6416,BigF+6420
        dc.l BigF+6424,BigF+6428,BigF+6432,BigF+6436



;***************************************************************************
; Ecco il testo: mettendo 1,2,3,4 si cambia la velocita' di scroll
;***************************************************************************

scrollmsg:
 dc.b 4,"AMIGA EXPERT TEAM",1,"        ",3
 dc.b " IL NUOVO GRUPPO ITALIANO DI UTENTI AMIGA EVOLUTI  ",2
 dc.b "       ",3
 dc.b "  SE VUOI METTERTI IN CONTATTO CON APPASSIONATI DI AMIGA ",2
```

```
 dc.b "CHE LO USANO PER FARCI MUSICA, GRAFICA, PROGRAMMAZIONE O ALTRO,"
 dc.b " SIA PER HOBBY CHE PER LAVORO, SCRIVI A: (MOUSE DESTRO PER STOP) ",1
 dc.b " MIRKO LALLI - VIA VECCHIA ARETINA 64 - 52020 LATERINA STAZIONE - ",2
 dc.b "AREZZO - ",3
 dc.b " CREDITI PER QUESTA DEMO: ",1
 dc.b "PROGRAMMAZIONE ASSEBLER E GRAFICA BY FABIO CIUCCI -",2
 dc.b " MUSICA PRESA DA UNA LIBRERIA PD ",3
 dc.b "-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-",4
 dc.b "=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-=+=-"
 dc.b "                                                        "
textend:

; Nota: Altro CLUB per Amiga e' APU: per informazioni tel. 081/5700434
;                                                           081/7314158
; giovedi'-venerdi' ore 19-22

*****************************************************************************
;               COPPERLIST:
*****************************************************************************

        section copper,data_c

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,$24        ; BplCon2 - Tutti gli sprite sopra i bitplane
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $100,$200       ; BplCon0 - no bitplanes

        dc.w    $0180,$000      ; color0 - SFONDO
        dc.w    $0182,$1af      ; color1 - SCRITTE

        dc.w    $9707,$FFFE     ; WAIT - disegna la barretta in alto
        dc.w    $180,$110       ; Color0
        dc.w    $9807,$FFFE     ; wait....
        dc.w    $180,$120
        dc.w    $9a07,$FFFE
        dc.w    $180,$130
        dc.w    $9b07,$FFFE
        dc.w    $180,$240
        dc.w    $9c07,$FFFE
        dc.w    $180,$250
        dc.w    $9d07,$FFFE
        dc.w    $180,$370
        dc.w    $9e07,$FFFE
        dc.w    $180,$390
        dc.w    $9f07,$FFFE
        dc.w    $180,$4b0
        dc.w    $a007,$FFFE
        dc.w    $180,$5d0
        dc.w    $a107,$FFFE
        dc.w    $180,$4a0
        dc.w    $a207,$FFFE
        dc.w    $180,$380
        dc.w    $a307,$FFFE
        dc.w    $180,$360
        dc.w    $a407,$FFFE
        dc.w    $180,$240
```

```
        dc.w    $a507,$FFFE
        dc.w    $180,$120
        dc.w    $a607,$FFFE
        dc.w    $180,$110

        dc.w    $A707,$FFFE
        dc.w    $180,$000

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo   bitplane
        dc.w $e4,0,$e6,0        ; secondo bitplane
        dc.w $e8,0,$ea,0        ; terzo   bitplane

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $108,4          ; bpl1mod - saltiamo i 4 bytes dove si
        dc.w    $10a,4          ; bpl2mod - vedrebbe stampare il testo...
                                ; Ricordatevi che lo schermo e' largo 44 bytes
                                ; in realta', per lasciare all'estrema destra,
                                ; fuori dal visibile, cio' che non deve essere
                                ; visto. Tutti gli scrolltext fanno cosi'.

        dc.w    $180,$000       ; colori
        dc.w    $182,$111
        dc.w    $184,$233
        dc.w    $186,$555
        dc.w    $188,$778
        dc.w    $18a,$aab
        dc.w    $18c,$fff
        dc.w    $18e,$fff

scol:
        DCB.w   6*64,0          ; spazio per le sfumature di colore generate
                                ; dalla routine "makecolors"

        dc.w    $EE07,$fffe
        dc.w    $180,$004

        dc.w    $184,$023,$186,$118             ; Colori piu' "blu"
        dc.w    $188,$25b,$18a,$38e,$18c,$acf

        dc.w    $182,$550       ; questa parte di copperlist
        dc.w    $18e,$155       ; realizza l'effetto specchio, dovreste sapere
        dc.w    $108,-84        ; in che modo!
        dc.w    $10A,-84
        dc.w    $F307,$fffe

        dc.w    $182,$440
        dc.w    $18e,$144
        dc.w    $108,-172
        dc.w    $10A,-172
        dc.w    $108,-84
        dc.w    $10A,-84
        dc.w    $180,$004
        dc.w    $F407,$fffe
        dc.w    $182,$330
        dc.w    $18e,$133
        dc.w    $108,-172
        dc.w    $10A,-172
        dc.w    $180,$005
        dc.w    $F607,$fffe
        dc.w    $182,$220
```

```
        dc.w    $18e,$123
        dc.w    $108,-84
        dc.w    $10A,-84
        dc.w    $180,$006
        dc.w    $FA07,$fffe
        dc.w    $182,$110
        dc.w    $18e,$012
        dc.w    $108,-172
        dc.w    $10A,-172
        dc.w    $180,$007
        dc.w    $FD07,$fffe
        dc.w    $182,$110
        dc.w    $18e,$011
        dc.w    $108,-84
        dc.w    $10A,-84
        dc.w    $180,$008
        dc.w    $ffdf,$fffe
        dc.w    $0107,$fffe
        dc.w    $0407,$fffe
        dc.w    $182,$001
        dc.w    $18e,$011
        dc.w    $108,-172
        dc.w    $10A,-172
        dc.w    $180,$009
        dc.w    $0607,$fffe
        dc.w    $182,$002
        dc.w    $18e,$111
        dc.w    $108,-84
        dc.w    $10A,-84
        dc.w    $180,$00A
        dc.w    $0A07,$fffe
        dc.w    $182,$003
        dc.w    $18e,$101
        dc.w    $108,-172
        dc.w    $10A,-172
        dc.w    $180,$00B
        dc.w    $0D07,$fffe
        dc.w    $182,$004
        dc.w    $18e,$202
        dc.w    $108,-84
        dc.w    $10A,-84
        dc.w    $180,$00e

        dc.w    $1307,$fffe
        dc.w    $100,$200       ; no bitplanes

        dc.w    $FFFF,$FFFE     ; Fine copperlist

;***************************************************************************

; Ecco il font, che sta in una immagine 320*200 a 3 bitplanes (8 colori)

BigF:
        incbin  "font4"

;***************************************************************************

        SECTION BUFY,BSS_C

BITPLANE:
        ds.b    3*44*256        ; spazio per 3 bitplanes
```

```
        END

In questo listato vediamo un'altro esempio di scrolltext, piu' sofisticato
del precedente.. Si tratta della routine di scroll usata nell'intro AMIGAET
di Fabio Ciucci. In questo programma, lo scrolltext si sposta in alto e in
basso. Per ottenere questo effetto vengono utilizzati 2 buffer per il testo.
Nel primo (invisibile) vengono stampati i caratteri e viene scrollato il testo.
Da qui il testo viene copiato nell'altro buffer (che e' quello visibile) ad
una posizione verticale che varia da un frame all'altro secondo una tabella.
Il secondo buffer non viene mai cancellato perche' durante la copia dal primo
buffer al secondo vengono copiate anche alcune righe "pulite" (azzerate) che
cancellano la parte del vecchio testo che non viene sovrascritta dalla
nuova. Per risparmiare memoria i 2 buffer sono stati riuniti in uno
(all'indirizzo BITPLANE) delle dimensioni di uno schermo 320*256 a 3 planes.
Cio' e' possibile perche' in realta' viene usato uno schermo alto solo 180
linee. Infatti la visualizzazione dei bitplanes viene attivata dalla copperlist
solo a partire dalla linea $A7 del display.
Un' altra particolarita' di questo listato e' che parte della copperlist
viene generata mediante una routine del processore, la "makecolors".
L'argomento delle copperlist generate dal processore (e dal blitter !)
verra' affrontato in una prossima lezione. Per il momento dategli comunque
uno sguardo.
```

Figura 25.10: Lezione 9n2

CAPITOLO 26

LEZIONE 10

26.1 Lezione10a1

```
; Lezione10a1.s BLITTATA, in cui disegnamo rettangoli sullo schermo
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #1-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
```

```
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; parametri per routine di disegno

        move.w  #16,d0                  ; X vertice superiore sinistro
        move.w  #10,d1                  ; Y vertice superiore sinistro
        move.w  #48,d2                  ; larghezza
        move.w  #20,d3                  ; altezza
        bsr.s   BlitRett                ; esegui la routine di disegno

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

; parametri per routine di disegno

        move.w  #64,d0                  ; X vertice superiore sinistro
        move.w  #70,d1                  ; Y vertice superiore sinistro
        move.w  #32,d2                  ; larghezza
        move.w  #40,d3                  ; altezza
        bsr.s   BlitRett                ; esegui la routine di disegno

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts

;*****************************************************************************
; Questa routine disegna un rettangolo sullo schermo.
;
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;*****************************************************************************

BlitRett:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

; calcolo indirizzo di partenza del blitter

        lea     bitplane1,a1    ; indirizzo bitplane
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a1           ; aggiungi ad indirizzo
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

; calcolo modulo blitter

        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d2,d4           ; modulo=larg. schermo-larg. rettangolo
```

```
; calcolo dimensione blittata

        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d2,d3           ; metti insieme le dimensioni

; carica i registri

        move.l  #$01ff0000,$40(a5)      ; BLTCON0 e BLTCON1
                                        ; usa il canale D
                                        ; LF=$FF (operaz. disegno)
                                        ; modo ascendente

        move.w  d4,$66(a5)              ; BLTDMOD
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$aaa      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        SECTION bitplane,BSS_C

BITPLANE1:
        ds.b    40*256

;***************************************************************************

        end
```

In questo esempio usiamo il blitter per tracciare dei rettangoli sullo schermo. Utilizziamo una routine parametrica, che traccia un rettangolo conoscendo le coordinate del vertice superiore sinistro e le dimensioni (larghezza e altezza) del rettangolo. Per semplificare la routine la larghezza e la posizione X del vertice sono approssimate a multipli di 16.
Il disegno viene realizzato mediante una blittata che pone l'uscita sempre a 1, che si ottiene ponendo LF=$FF come spiegato nella lezione.

Lezione10a2

```
; Lezione10a2.s BLITTATA, in cui copiamo un disegno invertendo un bitplane
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; copia l'immagine normalmente

        lea     FiguraPlane1,a0         ; copia il primo plane
        lea     BITPLANE1,a1
        bsr.s   copia

        lea     FiguraPlane2,a0         ; copia secondo plane
        lea     BITPLANE2,a1
        bsr.s   copia

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

; copia l'immagine invertendo il primo bitplane

        lea     FiguraPlane1,a0
        lea     BITPLANE1+14,a1
        bsr.s   CopiaInversa            ; copia il primo plane invertendolo

        lea     FiguraPlane2,a0
        lea     BITPLANE2+14,a1
```

```
        bsr.s   copia                   ; copia secondo plane normalmente

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:
        rts

;***************************************************************************
; Questa routine copia la figura sullo schermo.
; Prende come parametri
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;***************************************************************************

Copia:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*25)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
        rts                             ; altezza 25 linee

;***************************************************************************
; Questa routine copia la figura sullo schermo invertendola
; cioe‘ trasforma 1 in 0 e 0 in 1.
;
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;***************************************************************************

;                     _   _
;                   __/ \_/ \
;                  /  \_ oo_/
;                 /        \/_
;            ____/_ ___ ____o
;         ___/      \\  \\ UU

CopiaInversa:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$090f0000,$40(a5)      ; BLTCON0  e BLTCON1
                                        ; copia invertendo i bit, cioe‘
                                        ; D=NOT A
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*25)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
```

```
        rts                             ; altezza 25 linee

;****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$2200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane
        dc.w    $e4,$0000,$e6,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$aaa      ; color1
        dc.w    $0184,$55f      ; color2
        dc.w    $0186,$f80      ; color3

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;****************************************************************************

FiguraPlane1:
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$c000,$0000,$0003,$c000
        dc.w    $0000,$0003,$c000,$0000,$0003,$c000,$0000,$0003,$c000,$0000
        dc.w    $0003,$c000,$0000,$0003,$c000,$0000,$0003,$c000,$0000,$0003
        dc.w    $c25c,$3bbb,$bb83,$c354,$22aa,$a283,$c2d4,$22bb,$b303,$c254
        dc.w    $22a2,$2283,$c25c,$3ba2,$3a83,$c000,$0000,$0003,$c000,$0000
        dc.w    $0003,$c000,$0000,$0003,$c000,$0000,$0003,$c000,$0000,$0003
        dc.w    $c000,$0000,$0003,$c000,$0000,$0003,$c000,$0000,$0003,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff

FiguraPlane2:
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff
        dc.w    $ffff,$ffff,$ffff,$ffff,$ffff

;****************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256

        end

;****************************************************************************
```

```
In questo esempio vediamo un'applicazione dell'operazione logica di NOT.
Abbiamo un disegno sullo schermo, che potrebbe rappresentare un pulsante.
Supponiamo ora di voler disegnare lo stesso pulsante, ma invertendo i
colori, per simulare la pressione.
Un metodo e' quello di scambiare i colori nella copperlist.
In questo modo, pero' si scambiano i colori in tutto lo schermo e quindi se
vogliamo avere contemporaneamente 2 pulsanti, uno con i colori normali e
l'altro con i colori invertiti, questa tecnica non va bene.
Allora non ci resta che modificare i bitplane che formano l'immagine.
Il pulsante e' disegnato con i colori 2 e 3.
Per scambiare i colori dobbiamo trasformare il colore 2 in 3 e viceversa.
I pixel colorati con il colore 2:
Si ha il colore 2 quando il plane 1 e' settato a 0 e il plane 2 e' settato a 1.
Si ha il colore 3 quando il plane 1 e' settato a 1 e il plane 2 e' settato a 1.
Poiche' entrambi i colori hanno il plane 2 settato a 1, dobbiamo cambiare
solo il plane 1.
Se nel plane 1 invertiamo tutti i bit (cioe' trasformiamo tutti gli 0 in 1
e tutti gli 1 in 0) scambieremo i colori 2 e 3.
L'inversione dei bit e' l'operazione logica NOT che possiamo realizzare con il
blitter mediante un opportuno minterm. Se per leggere usiamo il canale A,
dobbiamo porre l'uscita D a 1 ogni volta che l'ingresso vale 0 e viceversa.
Cio' si ottiene settando a 1 tutti i minterms che corrispondono a combinazioni
con A=0, ovvero (come potete verificare dalla tabella nella lezione) con
LF=$0F.
```

Figura 26.1: Lezione 10a2

Lezione10b1

```
; Lezione9b1.s  Esempio di OR tra 2 canali
;               Tasto destro per eseguire la blittata, sinistro per uscire.
```

```
        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #1-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     Figura1,a0
        lea     BITPLANE1,a1
        bsr.s   copia           ; esegui copia figura 1

        lea     Figura2,a0
        lea     BITPLANE1+20,a1
        bsr.s   copia           ; esegui copia figura 2

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, non cancellare

        bsr.s   BlitOR          ; esegui l'OR tra le 2 figure

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:
        rts


;*****************************************************************************
; Questa routine copia la figura sullo schermo.
; Prende come parametri
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;*****************************************************************************

Copia:
```

```
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #30,$66(a5)             ; BLTDMOD (40-10=30)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*71)+5,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 5 word
        rts                             ; altezza 71 linee

;***************************************************************************
; Questa routine l'OR tra 2 figura lette attraverso i canali A e B
;***************************************************************************

;                  /#\    ...
;                 /   \  :   :
;                / /\  \c o o ø
;               /%/  \  (  ^  )    /)OO
;              (  u  / __\ O / \   \)(/
;              UUU_ ( /)  '-''  \  /%/
;               /  \| /   <  :\  )/ /
;              /  . \::.   >.( \ ' /
;             /  /\   ':::./|. ) \#/
;            /  /  \    ': ). )
;         __ û%,/    \   / (.  )
;        (  \% /     /  /  ) .'
;         \_ò /     /  /   ':'
;          \_/     /  /
;                 /\./
;                /.%
;               / %
;              (  %
;               \ ~\
;                \__)

BlitOR:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$0dfc0000,$40(a5)      ; BLTCON0 e BLTCON1
                                        ; usa i canali A,B e D
                                        ; esegue l'OR tra A e B (LF=$FC)
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #0,$62(a5)              ; BLTBMOD (=0)
        move.w  #30,$66(a5)             ; BLTDMOD (40-10=30)

        move.l  #Figura1,$50(a5)                ; BLTBPT  puntatore sorgente
        move.l  #Figura2,$4c(a5)                ; BLTAPT  puntatore sorgente
        move.l  #BITPLANE1+100*40+10,$54(a5)    ; BLTDPT  puntatore dest.
        move.w  #(64*71)+5,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 5 word
        rts                             ; altezza 71 linee
```

```
;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$aaa      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

Figura1:
        dc.w    $ffc0,0,0,$0007,$fe00,$8000,0,$1000,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$7c00,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,$0001,$ff00,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$0007,$ffc0,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$001f,$fff0,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$007f,$fffc,0,$0200,$8000,$007f,$fffc,0,$0200
        dc.w    $8000,$007f,$fffc,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$001f,$fff0,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$0007,$ffc0,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,$0001,$ff00,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$7c00,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$1000,0,$0200
```

```
        dc.w    $ffc0,0,0,$0007,$fe00

Figura2:
        dc.w    $ffff,$ffff,$ffff,$ffff,$fe00,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,$3800,0,0
        dc.w    0,$0003,$ff80,0,0,0,$001f,$fff0,0,0
        dc.w    0,$01ff,$ffff,0,0,0,$0fff,$ffff,$e000,0
        dc.w    0,$ffff,$ffff,$fe00,0,$0007,$ffff,$ffff,$ffc0,0
        dc.w    $007f,$ffff,$ffff,$fffc,0,$03ff,$ffff,$ffff,$ffff,$8000
        dc.w    $3fff,$ffff,$ffff,$ffff,$f800,$7fff,$ffff,$ffff,$ffff,$fc00
        dc.w    $3fff,$ffff,$ffff,$ffff,$f800,$03ff,$ffff,$ffff,$ffff,$8000
        dc.w    $007f,$ffff,$ffff,$fffc,0,$0007,$ffff,$ffff,$ffc0,0
        dc.w    0,$ffff,$ffff,$fe00,0,0,$0fff,$ffff,$e000,0
        dc.w    0,$01ff,$ffff,0,0,0,$001f,$fff0,0,0
        dc.w    0,$0003,$ff80,0,0,0,0,$3800,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $ffff,$ffff,$ffff,$ffff,$fe00

;******************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256

        end

;******************************************************************************

In questo esempio vediamo l'OR tra 2 figure. Si tratta di una semplice
blittata che esegue l'OR tra i 2 canali A e B, usando il valore di LF
calcolato nella lezione. Per esercizio potete modificarlo utilizzando
per la lettura il canale C al posto del B. Le modifiche da fare sono
le seguenti:
Sostiuire i registri modulo e puntatore del canale B con quelli del C;
attivare il canale C invece che il B;
Calcolare il giusto valore di LF per eseguira l'OR tra A e C.
Il calcolo di LF e' semplice: basta osservare la tabella in fig.27 e
settare ad 1 tutti i minterms corrispondenti a combinazioni con A=1 oppure
con C=1. Si ottiene LF=$FA.
Ripetete lo stesso esercizio per fare l'OR tra i canali B e C.
```

Lezione10b2

```
; Lezione10b2.s Esempio di AND tra i canali A e B
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #1-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     Figura1,a0
        lea     BITPLANE1,a1
        bsr.s   copia           ; esegui copia figura 1

        lea     Figura2,a0
        lea     BITPLANE1+20,a1
        bsr.s   copia           ; esegui copia figura 2

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

        bsr.s   BlitAND         ; esegui la routine di AND

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts


;*****************************************************************************
```

```
; Questa routine copia la figura sullo schermo.
; Prende come parametri
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;***************************************************************************


Copia:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #30,$66(a5)             ; BLTDMOD (40-10=30)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*71)+5,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 5 word
        rts                             ; altezza 71 linee

;***************************************************************************
; Questa routine effettua l'AND di 2 figure sullo schermo
;***************************************************************************

;               _*_
;              /   \
;             (_____)
;             o_o  '\
;            (___,   )
;            _\_____/__
;           /          \
;          /  .         \
;         /  /      Y    |
;         X_/       |____|
;         \(________/  /
;          \_/      \_/\
;        __.\   Y      /
;        )_| \  |     /
;       .---. \_|____/
;       |TaG|_/o|o  %\_
;       |___|___|______)

BlitAND:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$0dc00000,$40(a5)      ; BLTCON0  e BLTCON1
                                        ; usa A, B e D
                                        ; esegue un AND tra A e B (LF=$C0)
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #0,$62(a5)              ; BLTBMOD (=0)
        move.w  #30,$66(a5)             ; BLTDMOD (40-10=30)

        move.l  #Figura1,$50(a5)                ; BLTAPT puntatore sorgente
        move.l  #Figura2,$4c(a5)                ; BLTBPT puntatore sorgente
```

```
        move.l  #BITPLANE1+100*40+10,$54(a5)    ; BLTDPT puntatore destinazione
        move.w  #(64*71)+5,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 5 word
        rts                             ; altezza 71 linee

;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$aaa      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

Figura1:
        dc.w    $ffc0,0,0,$0007,$fe00,$8000,0,$1000,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$7c00,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,$0001,$ff00,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$0007,$ffc0,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$001f,$fff0,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$007f,$fffc,0,$0200,$8000,$007f,$fffc,0,$0200
        dc.w    $8000,$007f,$fffc,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$003f,$fff8,0,$0200
        dc.w    $8000,$003f,$fff8,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$001f,$fff0,0,$0200,$8000,$001f,$fff0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$000f,$ffe0,0,$0200,$8000,$000f,$ffe0,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$0007,$ffc0,0,$0200
        dc.w    $8000,$0007,$ffc0,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0003,$ff80,0,$0200
        dc.w    $8000,$0003,$ff80,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,$0001,$ff00,0,$0200,$8000,$0001,$ff00,0,$0200
        dc.w    $8000,0,$fe00,0,$0200,$8000,0,$fe00,0,$0200
```

```
        dc.w    $8000,0,$fe00,0,$0200,$8000,0,$fe00,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$7c00,0,$0200
        dc.w    $8000,0,$7c00,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$3800,0,$0200
        dc.w    $8000,0,$3800,0,$0200,$8000,0,$1000,0,$0200
        dc.w    $ffc0,0,0,$0007,$fe00

Figura2:
        dc.w    $ffff,$ffff,$ffff,$ffff,$fe00,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,$3800,0,0
        dc.w    0,$0003,$ff80,0,0,0,$001f,$fff0,0,0
        dc.w    0,$01ff,$ffff,0,0,0,$0fff,$ffff,$e000,0
        dc.w    0,$ffff,$ffff,$fe00,0,$0007,$ffff,$ffff,$ffc0,0
        dc.w    $007f,$ffff,$ffff,$fffc,0,$03ff,$ffff,$ffff,$ffff,$8000
        dc.w    $3fff,$ffff,$ffff,$ffff,$f800,$7fff,$ffff,$ffff,$ffff,$fc00
        dc.w    $3fff,$ffff,$ffff,$ffff,$f800,$03ff,$ffff,$ffff,$ffff,$8000
        dc.w    $007f,$ffff,$ffff,$fffc,0,$0007,$ffff,$ffff,$ffc0,0
        dc.w    0,$ffff,$ffff,$fe00,0,0,$0fff,$ffff,$e000,0
        dc.w    0,$01ff,$ffff,0,0,0,$001f,$fff0,0,0
        dc.w    0,$0003,$ff80,0,0,0,0,$3800,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $ffff,$ffff,$ffff,$ffff,$fe00

;******************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256

        end

;******************************************************************************

In questo esempio eseguiamo un AND tra 2 figure. L'esempio e' identico
a lezione10b1.s, con l'unica differenza di un diverso valore di LF.
Il valore di LF e' calcolato settando a 1 i minterms che corrispondono a
combinazioni degli ingressi con A=1 e contemporaneamente B=1.
Si tratta di 2 sole combinazioni, che danno luogo a LF=$C0.
```

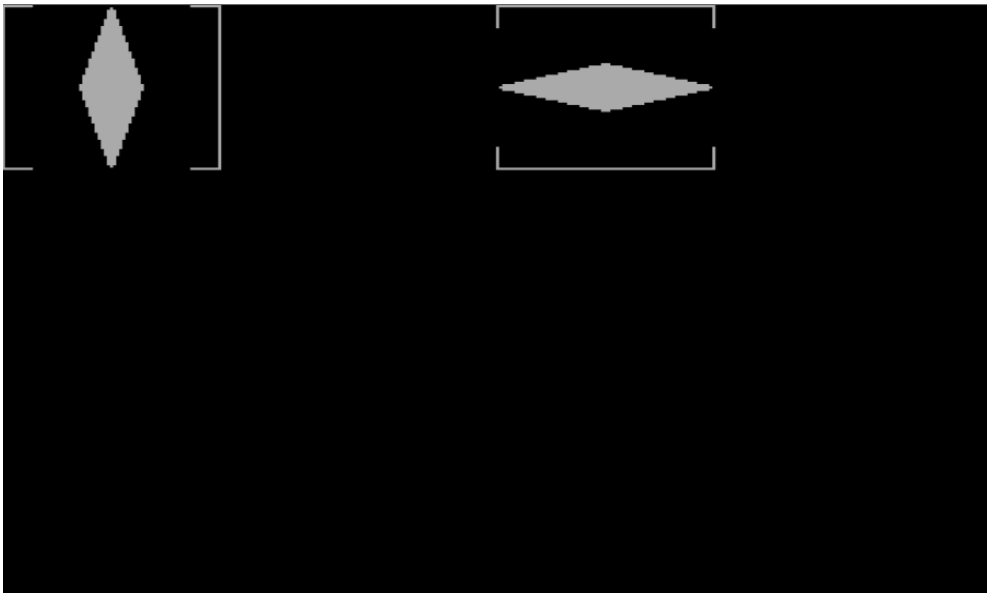


Figura 26.2: Lezione 10b2

26.2 Lezione10c1

```
; Lezione10c1.s BLITTATA, in cui costruiamo la maschera di un disegno
;               Alternare i tasti del mouse per vedere le blittate

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
```

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; copia l'immagine normalmente

        lea     FiguraPlane1,a0         ; copia il primo plane
        lea     BITPLANE1,a1
        bsr.s   copia

        lea     FiguraPlane2,a0         ; copia secondo plane
        lea     BITPLANE2,a1
        bsr.s   copia

        lea     FiguraPlane3,a0         ; copia terzo plane
        lea     BITPLANE3,a1
        bsr.s   copia

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

; copia primo bitplane

        lea     FiguraPlane1,a0
        lea     BITPLANE1+14,a1
        bsr.s   BlitOR          ; esegue un OR tra il plane 1 della figura
                                ; e la destinazione (vuota)

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        lea     FiguraPlane2,a0
        lea     BITPLANE1+14,a1
        bsr.s   BlitOR          ; esegue un OR tra il plane 2 della figura
                                ; e la destinazione (plane 1 della figura)
mouse3:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse3

        lea     FiguraPlane3,a0
        lea     BITPLANE1+14,a1
        bsr.s   BlitOR          ; esegue un OR tra il plane 3 della figura
                                ; e la destinazione (plane 1 OR 2 della figura)
mouse4:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse4
        rts

;******************************************************************************
; Questa routine copia la figura sullo schermo.
;
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;******************************************************************************

Copia:
        btst    #6,2(a5) ; dmaconr
```

```
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*42)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
        rts                             ; altezza 42 linee

;***************************************************************************
; Questa routine esegue un OR tra la sorgente e la destinazione.
; Usa i canali B,C e D. La sorgente viene letta attraverso il canale C.
; La destinazione invece viene letta dal canale B e poi riscritta tramite il D.
; Per conseguenza i canali B e D hanno lo stesso modulo e gli stessi indirizzi
; di partenza.
;
; Parametri:
;
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;***************************************************************************

BlitOR:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$07EE0000,$40(a5)      ; BLTCON0 e BLTCON1
                                        ; esegue un OR tra B e C
                                        ; D=B OR C
        move.w  #0,$60(a5)              ; BLTCMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.w  #34,$62(a5)             ; BLTBMOD (40-6=34)
        move.l  a0,$48(a5)              ; BLTCPT  puntatore sorgente
        move.l  a1,$4c(a5)              ; BLTBPT  puntatore destinazione
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*42)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
        rts                             ; altezza 42 linee

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$3200      ; bplcon0
```

```
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $180,$000       ; color0
        dc.w    $182,$aaa       ; color1
        dc.w    $184,$b00       ; color2
        dc.w    $186,$080       ; color3
        dc.w    $188,$24c
        dc.w    $18a,$eb0
        dc.w    $18c,$b52
        dc.w    $18e,$0cc

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questa e' la figura

FiguraPlane1:
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,0

FiguraPlane2:
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0

FiguraPlane3:
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$ffffff80
        dc.l    $ffffffff,$ff80ffff,$ffffff80,$ffffffff,$ff80ffff,$ffffff80
        dc.l    $f000ffff,$ff80f000,$ffffff80,$f000ffff,$ff80f000,$ffffff80
        dc.l    $f000ffff,$ff80f000,$ffffff80,$f000ffff,$ff80f000,$ffffff80
        dc.l    $f000ffff,$ff80f000,$ffffff80,$f000ffff,$ff80f000,$ffffff80
        dc.l    $ffffffff,$ff800000,0

;***************************************************************************
```

```
        SECTION bitplane,BSS_C

BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256

        end

;***************************************************************************
```

In questo esempio costruiamo con il blitter la maschera di una figura, ovverosia la sua "ombra". Per farlo e‘ necessario eseguire un OR tra i bit planes della figura. In questo esempio eseguiamo questa operazione un passo alla volta. Per prima cosa facciamo l’OR tra il primo bitplane della figura e la destinazione nella quale disegneremo la maschera. Poiche‘ all’inizio la destinazione e‘ vuota questo passo e‘ equivalente ad una semplice copia del primo plane della figura. Come secondo passo eseguiamo l’OR tra il secondo plane e la destinazione. Poiche‘ la destinazione contiene il primo plane, in pratica eseguiamo l’OR tra il plane 1 e il plane 2. Come terzo passo eseguiamo l’OR tra il plane 3 e la destinazione. Poiche‘ la destinazione contiene l’OR di plane 1 e plane 2, come risultato otterremo l’OR di tutti e 3 i planes. Se avessimo avuto una figura con piu‘ di 3 planes, avremmo dovuto ripetere questo stesso procedimento anche per gli altri planes. La blittata avviene utilizzando 3 canali. I planes della figura vengono letti attraverso il canale C. La destinazione invece viene letta attraverso il canale B e poi riscritta attraverso il canale D. Il valore di LF e‘ calcolato per effettuare l’OR dei canali B e C.

Lezione10c2

```
; Lezione10c2.s BLITTATA, in cui costruiamo la maschera di un disegno
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e’ alto 256 linee)
        addq.w  #8,a1
```

```
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; copia l'immagine normalmente

        lea     FiguraPlane1,a0         ; copia il primo plane
        lea     BITPLANE1,a1
        bsr.s   copia

        lea     FiguraPlane2,a0         ; copia secondo plane
        lea     BITPLANE2,a1
        bsr.s   copia

        lea     FiguraPlane3,a0         ; copia terzo plane
        lea     BITPLANE3,a1
        bsr.s   copia

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

; copia primo bitplane

        lea     FiguraPlane1,a0
        lea     FiguraPlane2,a1
        lea     FiguraPlane3,a2
        lea     BITPLANE1+14,a3
        bsr.s   BlitOR          ; esegue un OR tra i planes della figura
                                ; e copia il risultato
mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:
        rts

;*****************************************************************************
; Questa routine copia la figura sullo schermo.
;
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;*****************************************************************************

Copia:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*42)+3,$58(a5)      ; BLTSIZE (via al blitter !)
```

```
                                        ; larghezza 3 word
        rts                             ; altezza 42 linee

;***************************************************************************
; Questa routine esegue un OR tra i 3 canali A,B e C.
;
; A0 - indirizzo canale A
; A1 - indirizzo canale B
; A2 - indirizzo canale C
; A3 - indirizzo destinazione
;***************************************************************************

BlitOR:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$FFFFFFFF,$44(a5)      ; BLTFWM e BLTLWM
        move.l  #$0FFE0000,$40(a5)      ; BLTCON0 e BLTCON1
                                        ; attiva tutti i canali
                                        ; esegue un OR tra A,B e C
                                        ; D=A OR B OR C
        move.w  #0,$60(a5)              ; BLTCMOD (=0)
        move.w  #0,$62(a5)              ; BLTBMOD (=0)
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #34,$66(a5)             ; BLTDMOD (40-6=34)
        move.l  a0,$48(a5)              ; BLTCPT  puntatore sorgente
        move.l  a1,$4c(a5)              ; BLTBPT  puntatore destinazione
        move.l  a2,$50(a5)              ; BLTAPT  puntatore destinazione
        move.l  a3,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*42)+3,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 3 word
        rts                             ; altezza 42 linee

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$3200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $180,$000       ; color0
        dc.w    $182,$aaa       ; color1
        dc.w    $184,$b00       ; color2
        dc.w    $186,$080       ; color3
        dc.w    $188,$24c
        dc.w    $18a,$eb0
```

```
        dc.w    $18c,$b52
        dc.w    $18e,$0cc

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

; Questa e' la figura

FiguraPlane1:
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,0

FiguraPlane2:
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,$3fffff80
        dc.l    $3fff,$ff800000,$3fffff80,$3fff,$ff800000,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0

FiguraPlane3:
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    0,0,0,0,0,0
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$c0000000
        dc.l    $ffffc000,$ffff,$c0000000,$ffffc000,$ffff,$ffffff80
        dc.l    $ffffffff,$ff80ffff,$ffffff80,$ffffffff,$ff80ffff,$ffffff80
        dc.l    $f000ffff,$ff80f000,$ffffff80,$f000ffff,$ff80f000,$ffffff80
        dc.l    $f000ffff,$ff80f000,$ffffff80,$f000ffff,$ff80f000,$ffffff80
        dc.l    $f000ffff,$ff80f000,$ffffff80,$f000ffff,$ff80f000,$ffffff80
        dc.l    $ffffffff,$ff800000,0

;******************************************************************************

        SECTION bitplane,BSS_C

BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256

;******************************************************************************

        end
```

```
In questo esempio costruiamo con il blitter la maschera di una figura,
utilizzando una tecnica differente da quella impiegata in lezione10c1.s.
Questa volta infatti eseguiamo una sola blittata, usando tutti i canali.
Poiche‘ la nostra figura ha 3 bitplanes, possiamo leggerli contemporaneamente
attraverso i canali A,B e C e scrivere l’OR attraverso il canale D.
Questo e‘ il primo esempio che facciamo attivando tutti i canali del blitter
contemporaneamente. Notate, infatti, che i bit da 8 a 11 di BLTCON0 vengono
settati tutti a 1. Il valore di LF si calcola alla solita maniera, settando
cioe‘ a 1 tutti i minterms corrispondenti a combinazioni d’ingresso che abbiano
A=1 oppure B=1 oppure C=1. Naturalmente si tratta di 7 combinazioni
(l’unica non compresa e‘ quella con A=0, B=0 e C=0).
Notate che questa tecnica, a differenza di quella vista in lezione10c1.s
puo‘ essere applicata solo se la figura ha 3 bitplanes.
```

Lezione10c3

```
; Lezione10c3.s Effetto riflettore
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #FIGURA,d0      ; punta la figura
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e’ alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        move.l  #BITPLANE4,d0   ; punta il bitplane dove viene disegnata
        move.w  d0,6(a1)        ; la maschera
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l’AGA
        move.w  #$c00,$106(a5)          ; Disattiva l’AGA
        move.w  #$11,$10c(a5)           ; Disattiva l’AGA

mouse2:
```

```
Loop:
        cmp.b   #$ff,$6(a5)     ; VHPOSR - aspetta la linea $ff
        bne.s   loop
Aspetta:
        cmp.b   #$ff,$6(a5)     ; ancora linea $ff?
        beq.s   Aspetta

        bsr.s   ClearScreen     ; pulisci schermo
        bsr.w   SpostaMaschera  ; sposta posizione riflettore
        bsr.s   Riflettore      ; routine effetto

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts

;***************************************************************************
; Questa routine cancella la porzione di schermo interessata dalla blittata
;***************************************************************************

ClearScreen:
        lea     BITPLANE4+100*40,a1     ; indirizzo zona da cancellare (plane4)

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$01000000,$40(a5)      ; BLTCON0 + BLTCON1. Cancellazone
        move.w  #$0000,$66(a5)
        move.l  a1,$54(a5)
        move.w  #(64*39)+20,$58(a5)
        rts

;***************************************************************************
; Questa routine realizza l'effetto riflettore.
; Semplicemente la maschera viene disegnata sul bitplane 4
;***************************************************************************

;             ___________
;            /           \
;           /\            \
;          / /\____________)____
;          \/:/\___   ___/\     \
;           \/ ___ \_/ ___ \     \
;           ( /  o)   (  o\ )____/
;            \\__/ /Y\ \__//
;            (___/(_n_)\___)
;          __//\ _ _ _ _ /\\__
;         /==\\_Y Y Y Y Y_//==\
;        /    '-| | | | |-'    \
;       /       '-^-^-^-'       \

Riflettore:
        lea     BITPLANE4+100*40,a1     ; ind. destinazione

        move.w  MascheraX(PC),d0 ; posizione riflettore
        move.w  d0,d2           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
```

```
        lsl.w   #4,d0           ; della word...
        or.w    #$09F0,d0       ; ...giusti per inserirsi nel registro BLTCON0
                                ; notate LF=$F0 (cioe' copia A in D)
        lsr.w   #3,d2           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d2       ; escludo il bit 0 del
        add.w   d2,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.w  d0,$40(a5)              ; BLTCON0
        move.w  #$0000,$42(a5)          ; BLTCON1 modo ascendente
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #32,$66(a5)             ; BLTDMOD (40-8=32)

        move.l  #Maschera,$50(a5)       ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*39)+4,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 4 word
                                        ; altezza 39 linee

        rts

;***************************************************************************
; Questa routine legge la coordinata orizzontale da una tabella
; e la memorizza nella variabile MASCHERAX
;***************************************************************************

SpostaMaschera:
        ADDQ.L  #2,TABXPOINT            ; Fai puntare alla word successiva
        MOVE.L  TABXPOINT(PC),A0        ; indirizzo contenuto in long TABXPOINT
                                        ; copiato in a0
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BNE.S   NOBSTARTX               ; non ancora? allora continua
        MOVE.L  #TABX-2,TABXPOINT       ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        MOVE.W  (A0),MascheraX          ; copia il valore nella variabile
        rts

MascheraX:
                dc.w    0       ; posizione attuale maschera
TABXPOINT:
                dc.l    TABX    ; puntatore alla tabella

; tabella posizioni maschera

TABX:
        DC.W    $12,$16,$19,$1D,$21,$25,$28,$2C,$30,$34
        DC.W    $37,$3B,$3F,$43,$46,$4A,$4E,$51,$55,$58
        DC.W    $5C,$60,$63,$67,$6A,$6E,$71,$74,$78,$7B
        DC.W    $7F,$82,$85,$89,$8C,$8F,$92,$95,$98,$9C
        DC.W    $9F,$A2,$A5,$A8,$AA,$AD,$B0,$B3,$B6,$B8
        DC.W    $BB,$BE,$C0,$C3,$C5,$C8,$CA,$CC,$CF,$D1
```

```
        DC.W    $D3,$D5,$D8,$DA,$DC,$DE,$E0,$E1,$E3,$E5
        DC.W    $E7,$E8,$EA,$EC,$ED,$EE,$F0,$F1,$F2,$F4
        DC.W    $F5,$F6,$F7,$F8,$F9,$FA,$FB,$FB,$FC,$FD
        DC.W    $FD,$FE,$FE,$FF,$FF,$FF,$100,$100,$100,$100
        DC.W    $100,$100,$100,$100,$FF,$FF,$FF,$FE,$FE,$FD
        DC.W    $FD,$FC,$FB,$FB,$FA,$F9,$F8,$F7,$F6,$F5
        DC.W    $F4,$F2,$F1,$F0,$EE,$ED,$EC,$EA,$E8,$E7
        DC.W    $E5,$E3,$E1,$E0,$DE,$DC,$DA,$D8,$D5,$D3
        DC.W    $D1,$CF,$CC,$CA,$C8,$C5,$C3,$C0,$BE,$BB
        DC.W    $B8,$B6,$B3,$B0,$AD,$AA,$A8,$A5,$A2,$9F
        DC.W    $9C,$98,$95,$92,$8F,$8C,$89,$85,$82,$7F
        DC.W    $7B,$78,$74,$71,$6E,$6A,$67,$63,$60,$5C
        DC.W    $58,$55,$51,$4E,$4A,$46,$43,$3F,$3B,$37
        DC.W    $34,$30,$2C,$28,$25,$21,$1D,$19,$16,$12
FINETABX:

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$4200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000

Colours:
        dc.w    $0180,$000      ; colori da 0-7 tutti scuri. In questo modo
                                ; le parti della figura in corrispondenza delle
                                ; quali la maschera non e‘ disegnata sono
                                ; colorate in modo piu’ scuro
        dc.w    $0182,$011      ; color1
        dc.w    $0184,$223      ; color2
        dc.w    $0186,$122      ; color3
        dc.w    $0188,$112      ; color4
        dc.w    $018a,$011      ; color5
        dc.w    $018c,$112      ; color6
        dc.w    $018e,$011      ; color7

        dc.w    $0190,$000      ; colori 8-15 contengono la palette
        dc.w    $0192,$475
        dc.w    $0194,$fff
        dc.w    $0196,$ccc
        dc.w    $0198,$999
        dc.w    $019a,$232
        dc.w    $019c,$777
        dc.w    $019e,$444

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
;***************************************************************************

; qui c'e' il disegno, largo 320 pixel, alto 256 linee e formato da 3 plane

Figura:
        incbin  "amiga.raw"

;***************************************************************************

; Questa e' la maschera. E' una figura formata da un solo bitplane,
; alta 39 linee e larga 4 words

Maschera:
        dc.l    $00007fc0,$00000000,$0003fff8,$00000000,$000ffffe,$00000000
        dc.l    $001fffff,$00000000,$007fffff,$c0000000,$00ffffff,$e0000000
        dc.l    $01ffffff,$f0000000,$03ffffff,$f8000000,$03ffffff,$f8000000
        dc.l    $07ffffff,$fc000000,$0fffffff,$fe000000,$0fffffff,$fe000000
        dc.l    $1fffffff,$ff000000,$1fffffff,$ff000000,$1fffffff,$ff000000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $1fffffff,$ff000000,$1fffffff,$ff000000,$1fffffff,$ff000000
        dc.l    $0fffffff,$fe000000,$0fffffff,$fe000000,$07ffffff,$fc000000
        dc.l    $03ffffff,$f8000000,$03ffffff,$f8000000,$01ffffff,$f0000000
        dc.l    $00ffffff,$e0000000,$007fffff,$c0000000,$001fffff,$00000000
        dc.l    $000ffffe,$00000000,$0003fff8,$00000000,$00007fc0,$00000000

;***************************************************************************

; qui c'e' il quarto bitplane, dove viene disegnata la maschera.

        SECTION bitplane,BSS_C
BITPLANE4:
        ds.b    40*256

        end

;***************************************************************************
```

In questo esempio realizziamo un effetto "riflettore" con l'ausilio di un bitplane maschera. La tecnica e' la seguente. Abbiamo un disegno costituito da 3 bitplanes larghi 320 pixel e alti 256 linee. Per realizzare l'effetto, utilizziamo una "maschera", ovvero un disegno di un cerchio formato da 1 solo bitplane. Questa maschera viene disegnata e spostata su un quarto bitplane, come se fosse un bob. Siccome viene disegnata su un bitplane separato dalla figura, non dobbiamo preoccuparci dello sfondo. Si tratta sostanzialmente dello stesso trucco adottato in lezione9i4.s, l'esempio del bob con lo sfondo finto. Stavolta pero', per realizzare l'effetto riflettore, settiamo diversamente i colori nei registri: la palette della figura a 3 colori, cioe' i valori che normalmente scriveremmo nei registri COLOR00-COLOR07, vengono scritti stavolta nei registri COLOR08-COLOR15. Invece, i registri COLOR00-COLOR07 vengono tutti settati come piu' scuri (o neri). In questo modo, i pixel dell'immagine in corrispondenza dei quali il quarto bitplane e' settato a 0 appaiono tutti piu' scuri (avremmo potuto anche annerirli tutti); al contrario, i pixel dell'immagine in corrispondenza dei quali il quarto bitplane e' settato a 1 (cioe' in corrispondenza della maschera) appaiono con i giusti colori.
Questa tecnica e' molto veloce ma, analogamente all'esempio lezione9i4.s, ha uno svantaggio: utiliziamo 4 bitplane per un'immagine ad 8 colori.
Nel prossimo esempio vedremo come evitare questo problema.

Figura 26.3: Lezione 10c3

Lezione10c4

```
; Lezione10c4.s Effetto riflettore
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #5-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
```

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse2:

Loop:
        cmp.b   #$ff,$6(a5)     ; VHPOSR - aspetta la linea $ff
        bne.s   loop
Aspetta:
        cmp.b   #$ff,$6(a5)     ; ancora linea $ff?
        beq.s   Aspetta

        bsr.s   ClearScreen     ; pulisci schermo
        bsr.w   SpostaMaschera  ; sposta posizione riflettore
        bsr.s   Riflettore      ; routine effetto

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts

;******************************************************************************
; Questa routine cancella la porzione di schermo interessata dalla blittata
;******************************************************************************

ClearScreen:
        moveq   #5-1,d7                 ; 5 bit-planes
        lea     BITPLANE1+100*40,a1     ; indirizzo zona da cancellare (plane1)

ClearLoop:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$01000000,$40(a5)      ; BLTCON0 + BLTCON1. Cancellazone
        move.w  #$0000,$66(a5)
        move.l  a1,$54(a5)
        move.w  #(64*39)+20,$58(a5)
        add.l   #256*40,a1              ; indirizzo prossimo plane
        dbra    d7,Clearloop
        rts

;******************************************************************************
; Questa routine realizza l'effetto riflettore. Viene effettuata un'operazione
; di AND tra la figura e una maschera
;******************************************************************************

;             ___________
;            /           \
;           /\            \
;          / /\____________)____
;          \/:/\___   ___/\     \
;           \/ ___ \_/ ___ \     \
;           ( /  o)   (  o\ )____/
;            \\__/ /Y\ \__//
;            (___/(_n_)\___)
;          __//\ _ _ _ _ /\\__
```

```
;         /==\\_Y Y Y Y Y_//==\
;        /    '-| | | | |-'    \
;       /       '-^-^-^-'       \

Riflettore:
        moveq   #5-1,d7                 ; 5 bit-planes
        lea     Figura+40,a0            ; ind. figura
        lea     BITPLANE1+100*40,a1     ; ind. destinazione

        move.w  MascheraX(PC),d0 ; posizione riflettore
        move.w  d0,d2           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$0dc0,d0       ; ...giusti per inserirsi nel registro BLTCON0
                                ; notate LF=$C0 (cioe' AND tra A e B)
        lsr.w   #3,d2           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d2       ; escludo il bit 0 del
        add.w   d2,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto nella figura
        add.w   d2,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

Drawloop:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.w  d0,$40(a5)              ; BLTCON0
        move.w  #$0000,$42(a5)          ; BLTCON1 modo ascendente
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #32,$62(a5)             ; BLTBMOD (40-8=32)
        move.w  #32,$66(a5)             ; BLTDMOD (40-8=32)

        move.l  #Maschera,$50(a5)       ; BLTAPT  puntatore maschera
        move.l  a0,$4c(a5)              ; BLTBPT  puntatore figura
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*39)+4,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 4 word
                                        ; altezza 39 linee

        add.w   #56*40,a0               ; ind. prossimo plane figura
        add.w   #256*40,a1              ; ind. prossimo plane destinazione
        dbra    d7,Drawloop
        rts

;*****************************************************************************
; Questa routine legge la coordinata orizzontale da una tabella
; e la memorizza nella variabile MASCHERAX
;*****************************************************************************

SpostaMaschera:
        ADDQ.L  #2,TABXPOINT            ; Fai puntare alla word successiva
        MOVE.L  TABXPOINT(PC),A0        ; indirizzo contenuto in long TABXPOINT
```

```
                                        ; copiato in a0
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BNE.S   NOBSTARTX               ; non ancora? allora continua
        MOVE.L  #TABX-2,TABXPOINT       ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        MOVE.W  (A0),MascheraX          ; copia il valore nella variabile
        rts

MascheraX:
                dc.w    0       ; posizione attuale maschera
TABXPOINT:
                dc.l    TABX    ; puntatore alla tabella

; tabella posizioni maschera

TABX:
        DC.W    $12,$16,$19,$1D,$21,$25,$28,$2C,$30,$34
        DC.W    $37,$3B,$3F,$43,$46,$4A,$4E,$51,$55,$58
        DC.W    $5C,$60,$63,$67,$6A,$6E,$71,$74,$78,$7B
        DC.W    $7F,$82,$85,$89,$8C,$8F,$92,$95,$98,$9C
        DC.W    $9F,$A2,$A5,$A8,$AA,$AD,$B0,$B3,$B6,$B8
        DC.W    $BB,$BE,$C0,$C3,$C5,$C8,$CA,$CC,$CF,$D1
        DC.W    $D3,$D5,$D8,$DA,$DC,$DE,$E0,$E1,$E3,$E5
        DC.W    $E7,$E8,$EA,$EC,$ED,$EE,$F0,$F1,$F2,$F4
        DC.W    $F5,$F6,$F7,$F8,$F9,$FA,$FB,$FB,$FC,$FD
        DC.W    $FD,$FE,$FE,$FF,$FF,$FF,$100,$100,$100,$100
        DC.W    $100,$100,$100,$100,$FF,$FF,$FF,$FE,$FE,$FD
        DC.W    $FD,$FC,$FB,$FB,$FA,$F9,$F8,$F7,$F6,$F5
        DC.W    $F4,$F2,$F1,$F0,$EE,$ED,$EC,$EA,$E8,$E7
        DC.W    $E5,$E3,$E1,$E0,$DE,$DC,$DA,$D8,$D5,$D3
        DC.W    $D1,$CF,$CC,$CA,$C8,$C5,$C3,$C0,$BE,$BB
        DC.W    $B8,$B6,$B3,$B0,$AD,$AA,$A8,$A5,$A2,$9F
        DC.W    $9C,$98,$95,$92,$8F,$8C,$89,$85,$82,$7F
        DC.W    $7B,$78,$74,$71,$6E,$6A,$67,$63,$60,$5C
        DC.W    $58,$55,$51,$4E,$4A,$46,$43,$3F,$3B,$37
        DC.W    $34,$30,$2C,$28,$25,$21,$1D,$19,$16,$12
FINETABX:

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$5200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000
        dc.w $f0,$0000,$f2,$0000

Colours:
```

```
        dc.w    $180,0,$182,$f10,$184,$f21,$186,$f42
        dc.w    $188,$f53,$18a,$f63,$18c,$f74,$18e,$f85
        dc.w    $190,$f96,$192,$fa6,$194,$fb7,$196,$fb8
        dc.w    $198,$fc9,$19a,$f21,$19c,$f10,$19e,$f00
        dc.w    $1a0,$eff,$1a2,$eff,$1a4,$dff,$1a6,$dff
        dc.w    $1a8,$cff,$1aa,$bef,$1ac,$bef,$1ae,$adf
        dc.w    $1b0,$9df,$1b2,$9cf,$1b4,$8bf,$1b6,$7bf
        dc.w    $1b8,$7af,$1ba,$69f,$1bc,$68f,$1be,$57f

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; qui c'e' il disegno, largo 320 pixel, alto 56 linee e formato da 5 plane

Figura:
        incbin  lava320*56*5.raw

;***************************************************************************

; Questa e' la maschera. E' una figura formata da un solo bitplane,
; alta 39 linee e larga 4 words

Maschera:
        dc.l    $00007fc0,$00000000,$0003fff8,$00000000,$000ffffe,$00000000
        dc.l    $001fffff,$00000000,$007fffff,$c0000000,$00ffffff,$e0000000
        dc.l    $01ffffff,$f0000000,$03ffffff,$f8000000,$03ffffff,$f8000000
        dc.l    $07ffffff,$fc000000,$0fffffff,$fe000000,$0fffffff,$fe000000
        dc.l    $1fffffff,$ff000000,$1fffffff,$ff000000,$1fffffff,$ff000000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $1fffffff,$ff000000,$1fffffff,$ff000000,$1fffffff,$ff000000
        dc.l    $0fffffff,$fe000000,$0fffffff,$fe000000,$07ffffff,$fc000000
        dc.l    $03ffffff,$f8000000,$03ffffff,$f8000000,$01ffffff,$f0000000
        dc.l    $00ffffff,$e0000000,$007fffff,$c0000000,$001fffff,$00000000
        dc.l    $000ffffe,$00000000,$0003fff8,$00000000,$00007fc0,$00000000

;***************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256
BITPLANE4:
        ds.b    40*256
BITPLANE5:
        ds.b    40*256

        end

;***************************************************************************
```

Da notare che in questo esempio la grafica fuori dal "riflettore" deve essere per forza NERA, a differenza di Lezione10c3.s, dato che il sistema usato e' diverso, e soprattutto non ci sono abbastanza bitplanes.
In questo esempio vediamo un'altra tecnica per realizzare l'effetto "riflettore". Questa volta abbiamo un'immagine formata da 5 bitplane larghi 320 pixel e alti 56 linee memorizzata a partire dall'indirizzo "Figura".

Il fatto che l'immagine abbia 5 biplanes ci impedisce di usare la tecnica precedente, perche' non abbiamo nemmeno un bitplane libero per disegnarci separatamente la maschera.
Impieghiamo allora la seguente tecnica: eseguiamo un'operazione di AND tra la maschera e la figura, e scriviamo il risultato nei bitplanes dello schermo. In questo modo verranno copiati sullo schermo solo i bit della figura che corrispondono a un bit a 1 nella maschera. Poiche' la maschera e' a forma di cerchio, verra copiata una parte di figura a forma di cerchio.
Notate che siccome operiamo su uno schermo normale (non interleaved) dobbiamo blittare un plane per volta. Poiche' la maschera serve solo per selezionare i bit della figura, e' sempre la stessa per ogni bitplane.
L'operazione di AND e' naturalmente eseguita tramite il blitter. La maschera e la figura vengono lette attraverso i canali A e B. Il calcolo dei valori da scrivere nei registri (compresi i minterms) e' analogo a quelli effettuati nei precedenti esempi.

Lezione10c5

```
; Lezione10c5.s Effetto riflettore su schermo interleaved
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #5-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40,d0          ; INTERLEAVED! lungh. 1 linea
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse2:
Loop:
        cmp.b   #$ff,$6(a5)     ; VHPOSR - aspetta la linea $ff
        bne.s   loop
Aspetta:
```

```
        cmp.b   #$ff,$6(a5)     ; ancora linea $ff?
        beq.s   Aspetta

        bsr.s   ClearScreen     ; pulisci schermo
        bsr.w   SpostaMaschera  ; sposta posizione riflettore
        bsr.s   Riflettore      ; routine effetto

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:
        rts

;***************************************************************************
; Questa routine cancella la porzione di schermo interessata dalla blittata
;***************************************************************************

ClearScreen:
        lea     BITPLANE1+100*40*5,a1   ; indirizzo zona da cancellare (plane1)

        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$01000000,$40(a5)      ; BLTCON0 + BLTCON1. Cancellazone
        move.w  #$0000,$66(a5)
        move.l  a1,$54(a5)
        move.w  #(64*39*5)+20,$58(a5)
        rts

;***************************************************************************
; Questa routine realizza l'effetto riflettore. Viene effettuata un'operazione
; di AND tra la figura e una maschera
;***************************************************************************

;             ___________
;            /           \
;           /\            \
;          / /\____________)____
;          \/:/\___   ___/\     \
;           \/ ___ \_/ ___ \     \
;           ( /  o)   (  o\ )____/
;            \\__/ /Y\ \__//
;            (___/(_n_)\___)
;          __//\ _ _ _ _ /\\__
;         /==\\_Y Y Y Y Y_//==\
;        /    `-| | | | |-'    \
;       /       `-^-^-^-'       \

Riflettore:
        lea     Figura+40*5,a0          ; ind. figura
        lea     BITPLANE1+100*40*5,a1   ; ind. destinazione

        move.w  MascheraX(PC),d0 ; posizione riflettore
        move.w  d0,d2           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$0dc0,d0       ; ...giusti per inserirsi nel registro BLTCON0
                                ; notate LF=$C0 (cioe` AND tra A e B)
        lsr.w   #3,d2           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
```

```
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d2       ; escludo il bit 0 del
        add.w   d2,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto nella figura
        add.w   d2,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.w  d0,$40(a5)              ; BLTCON0
        move.w  #$0000,$42(a5)          ; BLTCON1 modo ascendente
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #32,$62(a5)             ; BLTBMOD (40-8=32)
        move.w  #32,$66(a5)             ; BLTDMOD (40-8=32)

        move.l  #Maschera,$50(a5)       ; BLTAPT  puntatore maschera
        move.l  a0,$4c(a5)              ; BLTBPT  puntatore figura
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*39*5)+4,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; larghezza 4 word
        rts                             ; altezza 39 linee
                                        ; 5 planes

;***************************************************************************
; Questa routine legge la coordinata orizzontale da una tabella
; e la memorizza nella variabile MASCHERAX
;***************************************************************************

SpostaMaschera:
        ADDQ.L  #2,TABXPOINT            ; Fai puntare alla word successiva
        MOVE.L  TABXPOINT(PC),A0        ; indirizzo contenuto in long TABXPOINT
                                        ; copiato in a0
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BNE.S   NOBSTARTX               ; non ancora? allora continua
        MOVE.L  #TABX-2,TABXPOINT       ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        MOVE.W  (A0),MascheraX          ; copia il valore nella variabile
        rts

MascheraX;
                dc.w    0       ; posizione attuale maschera
TABXPOINT:
                dc.l    TABX    ; puntatore alla tabella

; tabella posizioni maschera

TABX:
        DC.W    $12,$16,$19,$1D,$21,$25,$28,$2C,$30,$34
        DC.W    $37,$3B,$3F,$43,$46,$4A,$4E,$51,$55,$58
        DC.W    $5C,$60,$63,$67,$6A,$6E,$71,$74,$78,$7B
        DC.W    $7F,$82,$85,$89,$8C,$8F,$92,$95,$98,$9C
        DC.W    $9F,$A2,$A5,$A8,$AA,$AD,$B0,$B3,$B6,$B8
        DC.W    $BB,$BE,$C0,$C3,$C5,$C8,$CA,$CC,$CF,$D1
        DC.W    $D3,$D5,$D8,$DA,$DC,$DE,$E0,$E1,$E3,$E5
        DC.W    $E7,$E8,$EA,$EC,$ED,$EE,$F0,$F1,$F2,$F4
```

```
        DC.W    $F5,$F6,$F7,$F8,$F9,$FA,$FB,$FB,$FC,$FD
        DC.W    $FD,$FE,$FE,$FF,$FF,$FF,$100,$100,$100,$100
        DC.W    $100,$100,$100,$100,$FF,$FF,$FF,$FE,$FE,$FD
        DC.W    $FD,$FC,$FB,$FB,$FA,$F9,$F8,$F7,$F6,$F5
        DC.W    $F4,$F2,$F1,$F0,$EE,$ED,$EC,$EA,$E8,$E7
        DC.W    $E5,$E3,$E1,$E0,$DE,$DC,$DA,$D8,$D5,$D3
        DC.W    $D1,$CF,$CC,$CA,$C8,$C5,$C3,$C0,$BE,$BB
        DC.W    $B8,$B6,$B3,$B0,$AD,$AA,$A8,$A5,$A2,$9F
        DC.W    $9C,$98,$95,$92,$8F,$8C,$89,$85,$82,$7F
        DC.W    $7B,$78,$74,$71,$6E,$6A,$67,$63,$60,$5C
        DC.W    $58,$55,$51,$4E,$4A,$46,$43,$3F,$3B,$37
        DC.W    $34,$30,$2C,$28,$25,$21,$1D,$19,$16,$12
FINETABX:

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,160        ; VALORE MODULO = 2*20*(5-1)= 160
        dc.w    $10a,160

        dc.w    $100,$5200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000
        dc.w $f0,$0000,$f2,$0000

Colours:
        dc.w    $180,$000,$182,$f10,$184,$f21,$186,$f42
        dc.w    $188,$f53,$18a,$f63,$18c,$f74,$18e,$f85
        dc.w    $190,$f96,$192,$fa6,$194,$fb7,$196,$fb8
        dc.w    $198,$fc9,$19a,$f21,$19c,$f10,$19e,$f00
        dc.w    $1a0,$eff,$1a2,$eff,$1a4,$dff,$1a6,$dff
        dc.w    $1a8,$cff,$1aa,$bef,$1ac,$bef,$1ae,$adf
        dc.w    $1b0,$9df,$1b2,$9cf,$1b4,$8bf,$1b6,$7bf
        dc.w    $1b8,$7af,$1ba,$69f,$1bc,$68f,$1be,$57f

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; qui c'e' il disegno, largo 320 pixel, alto 56 linee e formato da 5 plane

Figura:
        incbin  lava320*56*5.rawblit

;***************************************************************************

; Questa e' la maschera. In formato interleaved e' formata da 5 planes
; alta 39 linee e larga 4 words

Maschera:
```

```
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $7fffff,$c0000000
        dc.l    $7fffff,$c0000000
        dc.l    $7fffff,$c0000000
        dc.l    $7fffff,$c0000000
        dc.l    $7fffff,$c0000000
        dc.l    $ffffff,$e0000000
        dc.l    $ffffff,$e0000000
        dc.l    $ffffff,$e0000000
        dc.l    $ffffff,$e0000000
        dc.l    $ffffff,$e0000000
        dc.l    $1ffffff,$f0000000
        dc.l    $1ffffff,$f0000000
        dc.l    $1ffffff,$f0000000
        dc.l    $1ffffff,$f0000000
        dc.l    $1ffffff,$f0000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $3ffffff,$f8000000
        dc.l    $7ffffff,$fc000000
        dc.l    $7ffffff,$fc000000
        dc.l    $7ffffff,$fc000000
        dc.l    $7ffffff,$fc000000
        dc.l    $7ffffff,$fc000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $fffffff,$fe000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
```

```
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$3fffffff,$ff800000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
	dc.l	$1fffffff,$ff000000
```

```
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $1fffffff,$ff000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $7fffffff,$fe000000
        dc.l    $3fffffff,$fc000000
        dc.l    $3fffffff,$fc000000
        dc.l    $3fffffff,$fc000000
        dc.l    $3fffffff,$fc000000
        dc.l    $3fffffff,$fc000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $1fffffff,$f8000000
        dc.l    $0fffffff,$f0000000
        dc.l    $0fffffff,$f0000000
        dc.l    $0fffffff,$f0000000
        dc.l    $0fffffff,$f0000000
        dc.l    $0fffffff,$f0000000
        dc.l    $07ffffff,$e0000000
        dc.l    $07ffffff,$e0000000
        dc.l    $07ffffff,$e0000000
        dc.l    $07ffffff,$e0000000
        dc.l    $07ffffff,$e0000000
        dc.l    $03ffffff,$c0000000
        dc.l    $03ffffff,$c0000000
        dc.l    $03ffffff,$c0000000
        dc.l    $03ffffff,$c0000000
        dc.l    $03ffffff,$c0000000
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $1fffff,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $ffffe,0
        dc.l    $3fff8,0
```

```
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $3fff8,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0
        dc.l    $7fc0,0

;***************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256
BITPLANE4:
        ds.b    40*256
BITPLANE5:
        ds.b    40*256

        end
```

In questo esempio realizziamo l'effetto "riflettore" con uno schermo interleaved. La tecnica e' la stessa di lezione10c4.s. L'unica differenza e' che stavolta, poiche' usiamo uno schermo interleaved, possiamo blittare i plane tutti insieme. Questo fatto (che come sappiamo aumenta la velocita') ci crea pero' problemi con la maschera. Infatti siccome blittiamo tutti i planes in una sola volta, abbiamo bisogno di una maschera per ogni plane. In questo caso quindi la nostra maschera e' formata da 5 planes tutti uguali. Poiche' e' interleaved, in pratica e' come prendere la maschera da un plane e ripetere ogni riga 5 volte. In questo caso, dunque il formato interleaved ci costringe a usare una maschera da 5 planes che occupa piu' memoria di quella da un solo plane.

26.3 Lezione10d1

```
; Lezione10d1.s BOB con ripristino dello sfondo.
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
```

```
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        bsr.w   LeggiMouse              ; leggi coordinate
        bsr.s   ControllaCoordinate     ; evita che il bob esca dallo schermo
        bsr.w   SalvaSfondo             ; salva lo sfondo
        bsr.s   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   RipristinaSfondo        ; ripristina lo sfondo

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts


;*****************************************************************************
; Questa routine fa in modo che le coordinate del bob rimangano sempre
; all'interno dello schermo.
;*****************************************************************************

ControllaCoordinate:
        tst.w   ogg_x           ; controlla X
        bpl.s   NoMinX          ; controlla bordo sinistro
        clr.w   ogg_x           ; se X e' negativa, pone X=0
        bra.s   controllaY      ; va a controllare la Y

NoMinX:
        cmp.w   #319-32,ogg_x   ; controlla il bordo destro. In X_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; sinistro del bob. Se esso ha raggiunto
                                ; 319-32, allora il bordo destro ha raggiunto
                                ; la coordinata 319
        bls.s   controllaY      ; se e' minore tutto bene, controlla la Y
        move.w  #319-32,ogg_x   ; altrimenti fissa la coordinata sul bordo.

controllaY:
        tst.w   ogg_y           ; controlla bordo in alto
```

```
        bpl.s   NoMinY          ; se e' positiva controlla in basso
        clr.w   ogg_y           ; altrimenti poni Y=0
        bra.s   EndControlla    ; ed esci

NoMinY:
        cmp.w   #255-11,ogg_y   ; controlla il bordo basso. In Y_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; alto del bob. Se esso ha raggiunto
                                ; Y=255-11, allora il bordo basso ha raggiunto
                                ; la coordinata Y=255
        bls.s   EndControlla    ; se e' minore tutto bene, controlla la Y
        move.w  #255-11,ogg_y   ; altrimenti fissa la coordinata sul bordo.
EndControlla:
        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato normale, e pertanto
; sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
; tecnica di mascherare l'ultima word del BOB vista nella lezione
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        move.w  d0,d2

        or.w    #$0FCA,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
```

```
        move.w  d2,$42(a5)              ; BLTCON1 (nessun modo speciale)
        move.l  #$0022fffe,$60(a5)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito
        move.l  #Maschera,$50(a5)       ; BLTAPT  (fisso alla maschera)
        move.l  a0,$54(a5)              ; BLTDPT  (schermo)
        move.l  a0,$48(a5)              ; BLTCPT  (schermo)
        move.l  a1,$4c(a5)              ; BLTBPT  (figura bob)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     4*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni plane e‘ largo 2 words e alto
                                        ; 11 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

;***************************************************************************
; Questa routine copia il rettangolo di sfondo che verra‘ sovrascritto dal
; BOB in un buffer
;***************************************************************************

SalvaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     Buffer,a1       ; indirizzo destinazione
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00220000,$64(a5)      ; BLTAMOD=40-4=36=$24
                                        ; BLTDMOD=0 nel buffer
        move.l  a0,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  a1,$54(a5)              ; BLTDPT - buffer
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)
```

```
        lea     40*256(a0),a0           ; punta al prossimo plane sorgente
        lea     6*11(a1),a1             ; punta al prossimo plane destinazione
                                        ; ogni blittata e' larga 3 words e alto
                                        ; 11 righe
        dbra    d7,PlaneLoop2

        rts

;***************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione.
;***************************************************************************

RipristinaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     Buffer,a1       ; indirizzo sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop3:
        btst    #6,2(a5)         ; dmaconr
WBlit4:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00000022,$64(a5)      ; BLTAMOD=0 (buffer)
                                        ; BLTDMOD=40-6=34=$22
        move.l  a1,$50(a5)              ; BLTAPT (buffer)
        move.l  a0,$54(a5)              ; BLTDPT (schermo)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        lea     6*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni blittata e' larga 3 words e alto
                                        ; 11 righe
        dbra    d7,PlaneLoop3
        rts

;***************************************************************************
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili OGG_X e OGG_Y
;***************************************************************************
```

```
LeggiMouse:
        move.b  $dff00a,d1      ; JOY0DAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_y        ; modifica posizione oggetto

no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_x        ; modifica pos. oggetto
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO = 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************
```

```
; Questi sono i dati che compongono la figura del bob.
; Il bob e‘ in formato normale, largo 32 pixel (2 words)
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000       ; plane 1
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $0ffffe00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000

        dc.l    $00000000       ; plane 2
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000
        dc.l    $00000000

        dc.l    $007fc000       ; plane 3
        dc.l    $03803800
        dc.l    $04000400
        dc.l    $081f8200
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $081f8200
        dc.l    $04000400
        dc.l    $03803800
        dc.l    $007fc000

Maschera:
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $0ffffe00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000



;******************************************************************************

BITPLANE:
        incbin  "amiga.raw"             ; qua carichiamo la figura

;******************************************************************************
```

```
        SECTION BUFFER,BSS_C

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 11, larghezza 3 words
; 3 bitplanes

Buffer:
        ds.w    11*3*3

        end

;***************************************************************************

In questo esempio risolviamo il problema dello sfondo con i BOB.
La struttura del programma e' la stessa di lezione9i3.s.
Le differenze sono tutte nella routine "DisegnaOggetto", che adotta la
procedura di disegno spiegata nella lezione. Come vedete per la blittata
si pone LF=$CA (cookie-cut) e si usano tutti i canali del blitter (A per la
maschera, B per il BOB e C per lo sfondo).
```

Lezione10d1r

```
; Lezione10d1r.s        BOB rawblit con ripristino dello sfondo.
;                       Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
                                ; QUI C'E' LA PRIMA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
```

```
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        bsr.w   LeggiMouse              ; leggi coordinate
        bsr.s   ControllaCoordinate     ; evita che il bob esca dallo schermo
        bsr.w   SalvaSfondo             ; salva lo sfondo
        bsr.s   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   RipristinaSfondo        ; ripristina lo sfondo

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts


;*****************************************************************************
; Questa routine fa in modo che le coordinate del bob rimangano sempre
; all'interno dello schermo.
;*****************************************************************************

ControllaCoordinate:
        tst.w   ogg_x           ; controlla X
        bpl.s   NoMinX          ; controlla bordo sinistro
        clr.w   ogg_x           ; se X e' negativa, pone X=0
        bra.s   controllaY      ; va a controllare la Y

NoMinX:
        cmp.w   #319-32,ogg_x   ; controlla il bordo destro. In X_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; sinistro del bob. Se esso ha raggiunto
                                ; 319-32, allora il bordo destro ha raggiunto
                                ; la coordinata 319
        bls.s   controllaY      ; se e' minore tutto bene, controlla la Y
        move.w  #319-32,ogg_x   ; altrimenti fissa la coordinata sul bordo.

controllaY:
        tst.w   ogg_y           ; controlla bordo in alto
        bpl.s   NoMinY          ; se e' positiva controlla in basso
        clr.w   ogg_y           ; altrimenti poni Y=0
        bra.s   EndControlla    ; ed esci

NoMinY:
        cmp.w   #255-11,ogg_y   ; controlla il bordo basso. In Y_OGG
                                ; e' memorizzata la coordinata del bordo
                                ; alto del bob. Se esso ha raggiunto
                                ; Y=255-11, allora il bordo basso ha raggiunto
                                ; la coordinata Y=255
        bls.s   EndControlla    ; se e' minore tutto bene, controlla la Y
        move.w  #255-11,ogg_y   ; altrimenti fissa la coordinata sul bordo.
EndControlla:
        rts
```

```
;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG. Il BOB e lo schermo sono in formato interleaved, e pertanto
; sono utilizzate le formule relative a questo formato nel calcolo dei
; valori da scrivere nei registri del blitter. Inoltre viene impiegata la
; tecnica di mascherare l'ultima word del BOB vista nella lezione
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        move.w  d0,d2

        or.w    #$0FCA,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  d2,$42(a5)              ; BLTCON1 (nessun modo speciale)
        move.l  #$0022fffe,$60(a5)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito

        move.l  #Maschera,$50(a5)       ; BLTAPT  (fisso alla maschera)
        move.l  a0,$54(a5)              ; BLTDPT  (schermo)
        move.l  a0,$48(a5)              ; BLTCPT  (schermo)
        move.l  #figura,$4c(a5)         ; BLTBPT  (figura bob)
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)

        rts

;***************************************************************************
; Questa routine copia il rettangolo di sfondo che verra‘ sovrascritto dal
; BOB in un buffer
;***************************************************************************
```

```
SalvaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione
        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00220000,$64(a5)      ; BLTAMOD=40-4=36=$24
                                        ; BLTDMOD=0 nel buffer
        move.l  a0,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  #Buffer,$54(a5)         ; BLTDPT - buffer
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)
        rts

;***************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione.
;***************************************************************************

RipristinaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        btst    #6,2(a5)         ; dmaconr
WBlit4:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
```

```
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00000022,$64(a5)      ; BLTAMOD=0 (buffer)
                                        ; BLTDMOD=40-6=34=$22
        move.l  #Buffer,$50(a5)         ; BLTAPT (buffer)
        move.l  a0,$54(a5)              ; BLTDPT (schermo)
        move.w  #(64*11*3)+3,$58(a5)    ; BLTSIZE (via al blitter !)
        rts

;***************************************************************************
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili OGG_X e OGG_Y
;***************************************************************************

LeggiMouse:
        move.b  $dff00a,d1      ; JOY0DAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_y        ; modifica posizione oggetto

no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,ogg_x        ; modifica pos. oggetto
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell’oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell’oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

                                ; QUI C’E‘ LA SECONDA DIFFERENZA RISPETTO
                                ; ALLE IMMAGINI NORMALI!!!!!!
        dc.w    $108,80         ; VALORE MODULO = 2*20*(3-1)= 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.
```

```
        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e‘ in formato interleaved, largo 32 pixel (2 words)
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000,$00000000,$007fc000
        dc.l    $03fff800,$007fc000,$03803800
        dc.l    $07fffc00,$03fff800,$04000400
        dc.l    $0ffffe00,$07fffc00,$081f8200
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $1fe07f00,$0fe07e00,$10204100
        dc.l    $0ffffe00,$07fffc00,$081f8200
        dc.l    $07fffc00,$03fff800,$04000400
        dc.l    $03fff800,$007fc000,$03803800
        dc.l    $007fc000,$00000000,$007fc000

Maschera:
        dc.l    $007fc000
        dc.l    $007fc000
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $03fff800
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $07fffc00
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $0ffffe00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $0ffffe00
        dc.l    $0ffffe00
        dc.l    $0ffffe00
```

```
        dc.l    $07fffc00
        dc.l    $07fffc00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $03fff800
        dc.l    $03fff800
        dc.l    $007fc000
        dc.l    $007fc000
        dc.l    $007fc000

;***************************************************************************

BITPLANE:
        incbin  "amiga.rawblit"         ; qua carichiamo la figura

;***************************************************************************

        SECTION BUFFER,BSS_C

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 11, larghezza 3 words
; 3 bitplanes

Buffer:
        ds.w    11*3*3

        end

;***************************************************************************

Questo esempio e' la versione rawblit di lezione10d1.s.
Confrontate le differenze nelle formule per il calcolo dei valori da scrivere
nei registri del blitter.
```

26.4 Lezione10e1

```
; Lezione10e1.s Blittata normale con copper monitor
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
```

```
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y

mouse:

        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0f400,d2      ; linea da aspettare = $F4
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $F4
        BNE.S   Waity1

;          \\\|||///
;         .  =======
;       / \| O   O |
;       \ / \`___'/
;        #   _| |_
;       (#) (     )
;        #\//|* *|\\
;        #\/(  *  )/
;        #   =====
;        #   ( U )
;        #   || ||
;       .#---'| |`----.
;       `#----' `-----'

        move.w  #$f00,$180(a5)          ; cambia il colore di sfondo
        bsr.s   DisegnaOggetto          ; disegna il bob

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)          ; rimetti lo sfondo nero

        bra.s   mouse

fine:
        rts


;*****************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
```

```
; X_OGG e Y_OGG.
;*****************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche’ vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l’ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=0
                                        ; BLTDMOD=40-36=4 come al solito

        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)

        lea     2*18*45(a1),a1          ; punta al prossimo plane sorgente
                                        ; ogni plane e‘ largo 18 words e alto
                                        ; 45 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell’oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell’oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse
```

```
;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e‘ in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.raw

;***************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;***************************************************************************

In questo esempio effettuiamo una copia mediante il blitter di un rettangolo
largo 18 words, alto 45 righe e formato da 3 bitplanes in formato normale
(quindi sono necessarie 3 blittate separate). Mediante il "copper monitor"
misuriamo la velocita‘ dell’operazione. La copia inizia quando il pennello
elettronico raggiunge la riga $f4. A questo punto, subito prima di iniziare
la copia cambiamo il COLOR 0. Quando la copia e‘ finita rimettiamo lo sfondo
al colore iniziale (nero).
Potete vedere mediante questa routine come la dimensione della blittata
```

```
influenzi la velocita‘: provate a cambiare la larghezza e/o l’altezza e
osservate i risultati. Comodo il "copper monitor", vero?
```

Figura 26.4: Lezione 10e1

Lezione10e1r

```
; Lezione10e1r.s        Blittata interleaved con copper monitor
;                       Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
```

```
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y

mouse:

        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0f400,d2      ; linea da aspettare = $F4, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $F4
        BNE.S   Waity1

;          \\\|||///
;         .  =======
;       / \| O   O |
;       \ / \'___'/
;        #   _| |_
;       (#) (     )
;        #\//|* *|\\
;        #\/(  *  )/
;        #   =====
;        #   ( U )
;        #   || ||
;       .#---'| |'----.
;       '#----' '-----'

        move.w  #$f00,$180(a5)          ; cambia il colore di sfondo
        bsr.s   DisegnaOggetto          ; disegna il bob

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)        ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)          ; rimetti lo sfondo nero

        bra.s   mouse

fine:
        rts


;*****************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;*****************************************************************************
```

```
DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche’ vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione


        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l’ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=0
                                        ; BLTDMOD=40-36=4 come al solito

        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(3*64*45)+18,$58(a5)   ; BLTSIZE (via al blitter !)

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell’oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell’oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
```

```
        dc.w    $108,80         ; VALORE MODULO 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e‘ in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.rawblit

;***************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;***************************************************************************

Questo programma e‘ la versione rawblit di lezione10e1.s.
La misura tiene conto solo del tempo impiegato dal blitter, e non evidenzia
i vantaggi del rawblit.
```

Lezione10e2

```
; Lezione10e2.s Cancellazione e disegno con copper monitor
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************
```

```
                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y

mouse:

        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0d000,d2      ; linea da aspettare = $D0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $D0
        BNE.S   Waity1

;          \\\|||///
;         .  =======
;       / \| O   O |
;       \ / \'___'/
;        #   _| |_
;       (#) (     )
;        #\//|* *|\\
;        #\/(  *  )/
;        #   =====
;        #   ( U )
;        #   || ||
;       .#---'| |'----.
;       '#----' '-----'

        bsr.s   CancellaOggetto         ; cancella il bob

        move    #$0b0,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito la cancellazione
```

```
        bsr.s   DisegnaOggetto          ; disegna il bob

        move    #$b00,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito il disegno

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)

        bra.s   mouse

fine:
        rts


;******************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob
;******************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.w  #$0f0,$180(a5)          ; coppermonitor! schermo verde durante
                                        ; la cancellazione.

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0004,$66(a5)          ; BLTDMOD=40-36=4
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
                                        ; il BOB

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop2

        rts
```

```
;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche’ vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.w  #$f00,$180(a5)

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l’ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=0
                                        ; BLTDMOD=40-36=4 come al solito
        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)

        lea     2*18*45(a1),a1          ; punta al prossimo plane sorgente
                                        ; ogni plane e‘ largo 18 words e alto
                                        ; 45 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell’oggetto
```

```
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.raw

;***************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;***************************************************************************
```

In questo esempio spostiamo un rettangolo sullo schermo, ovvero lo cancelliamo e lo ridisegnamo. Abbiamo utilizzato una serie di colori diversi per evidenziare la velocita' delle routine e il lavoro svolto dal blitter e dal processore. Quando inizia la routine di cancellazione, lo schermo diventa

di colore rosso chiaro. In questo momento, sia il blitter che il processore sono impegnati nell'operazione di cancellazione. Quando il processore termina la routine di cancellazione, il blitter non ha ancora finito di eseguire il suo compito. Il processore pero' e' gia' libero per altri compiti. Questa situazione e' evidenziata colorando lo schermo di rosso piu' scuro all'uscita della routine "CancellaOggetto". A questo punto il processore entra nella routine "DisegnaOggetto". Poiche' il blitter e' ancora impegnato prima che l'operazione di disegno abbia inizio e' necessario attendere il blitter. Quando finalmente il blitter si libera lo schermo viene colorato di verde chiaro. All'uscita della routine "DisegnaOggetto" lo schermo viene colorato di verde scuro in attesa che il blitter finisca di disegnare. Quando cio' accade, lo schermo ridiventa nero.

Notate inoltre che la cancellazione e la copia richiedono (circa) la stessa velocita'.

Lezione10e2r

```
; Lezione10e2r.s        Cancellazione e disegno rawblit con copper monitor
;                       Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40,d0          ; + LUNGHEZZA DI UNA RIGA !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y

mouse:
```

```
        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0d000,d2      ; linea da aspettare = $D0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $D0
        BNE.S   Waity1

;          \\\|||///
;         .  =======
;       / \| O   O |
;       \ / \'___'/
;        #   _| |_
;       (#) (     )
;        #\//|* *|\\
;        #\/(  *  )/
;        #   =====
;        #   ( U )
;        #   || ||
;       .#---'| |'----.
;       '#----' '-----'

        bsr.s   CancellaOggetto         ; cancella il bob

        move    #$0b0,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito la cancellazione

        bsr.s   DisegnaOggetto          ; disegna il bob

        move    #$b00,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito il disegno

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)

        bra.s   mouse

fine:
        rts


;*****************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob
;*****************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e' costituita da
                                ; 3 planes di 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
```

```
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.w  #$0f0,$180(a5)          ; coppermonitor! schermo verde durante
                                        ; la cancellazione.

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0004,$66(a5)          ; BLTDMOD=40-36=4
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(3*64*45)+18,$58(a5)   ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
                                        ; il BOB

        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #3*40,d0        ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 3 planes di 40 bytes ciascuno
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione


        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2
```

```
        move.w  #$f00,$180(a5)

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=0
                                        ; BLTDMOD=40-36=4 come al solito

        move.l  #figura,$50(a5)         ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(3*64*45)+18,$58(a5)   ; BLTSIZE (via al blitter !)

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,80         ; VALORE MODULO 80
        dc.w    $10a,80         ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e‘ in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.rawblit
```

```
;***************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;***************************************************************************

Questo programma e‘ la versione rawblit di lezione10e2.s.
Questa volta il copper monitor ci evidenzia anche il tempo impiegato dal
processore evidenziando in questo modo i vantaggi del rawblit.
```

Lezione10e3

```
; Lezione10e3.s operazione a 3 canali con copper monitor
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y
```

```
mouse:

        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0d000,d2      ; linea da aspettare = $D0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $D0
        BNE.S   Waity1

;                __
;               /  \
;              / ..|\
;             (_\  |_)
;              /  \@'
;             /     \
;         _  /  `   |
;        \\/  \  | _\
;         \   /_ || \\_
;          \____)|_) \_)

        bsr.s   CancellaOggetto         ; cancella il bob

        move    #$0b0,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito la cancellazione

        bsr.s   DisegnaOggetto          ; disegna il bob

        move    #$b00,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito il disegno

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)

        bra.s   mouse

fine:
        rts


;*****************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob
;*****************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
```

```
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.w  #$0f0,$180(a5)          ; coppermonitor! schermo verde durante
                                        ; la cancellazione.

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0004,$66(a5)          ; BLTDMOD=40-36=4
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
                                        ; il BOB

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop2

        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$0dfc,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
```

```
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.w  #$f00,$180(a5)

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-36=4 come al solito
        move    #0,$62(a5)              ; BLTBMOD

        move.l  #plane_OR,$4c(a5)       ; BLTBPT
        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)

        lea     2*18*45(a1),a1          ; punta al prossimo plane sorgente
                                        ; ogni plane e' largo 18 words e alto
                                        ; 45 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
```

```
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.raw

;***************************************************************************

; questo e' il plane fittizio per l'or. Ha dimensioni pari al bob
; ed e' totalmente azzerato.

plane_OR:
        dcb.b   36*45,0

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;***************************************************************************

In questo esempio invece di fare una copia da A a D, facciamo un OR tra i
canali A e B e scriviamo il risultato in D. In pratica otteniamo sempre
la stessa cosa, perche' il canale B legge un plane formato da tutti 0,
ma questo esempio serve per mostrare come una blittata a 3 canali sia
piu' lenta che una a 2
```

Lezione10e4

```
; Lezione10e4.s Copia da B a D con copper monitor
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
```

```
        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y

mouse:

        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0d000,d2      ; linea da aspettare = $D0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $D0
        BNE.S   Waity1

;            __
;           >(. )
;            |  (     /)
;            |   \___/ )
;            (   ----- )  >@)_//   >@)_//  >@)_//  >@)_//
;             \_______/    (__)     (__)    (__)    (__)
;           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        bsr.s   CancellaOggetto         ; cancella il bob

        move    #$0b0,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito la cancellazione

        bsr.s   DisegnaOggetto          ; disegna il bob

        move    #$b00,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito il disegno

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)
```

```
        bra.s   mouse

fine:
        rts


;***************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob
;***************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.w  #$0f0,$180(a5)          ; coppermonitor! schermo verde durante
                                        ; la cancellazione.

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0004,$66(a5)          ; BLTDMOD=40-36=4
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
                                        ; il BOB

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop2

        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
```

```
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.w  #$f00,$180(a5)

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  #$05CC,$40(a5)          ; BLTCON0 (usa B+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.w  #$0000,$62(a5)          ; BLTBMOD=0
        move.w  #$0004,$66(a5)          ; BLTDMOD=40-36=4 come al solito

        move.l  a1,$4c(a5)              ; BLTBPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)

        lea     2*18*45(a1),a1          ; punta al prossimo plane sorgente
                                        ; ogni plane e' largo 18 words e alto
                                        ; 45 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
```

```
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.raw

;*****************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;*****************************************************************************

In questo programma eseguiamo il disegno mediante una copia da B a D.
In questo caso, a differenza del caso in cui la copia era tra A e D, la
routine di disegno e' piu' lenta di quella di cancellazione.
```

Lezione10e5

```
; Lezione10e5.s Cancellazione e disegno con Blitter Nasty
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE_C

; Il codice va in memoria CHIP per mostrare la differenza con il caso
; normale

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
```

```
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
                                        ; Blitter Nasty ON


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #0,ogg_x
        move.w  #0,ogg_y

mouse:

        addq.w  #1,ogg_y
        cmp.w   #130,ogg_y
        beq.s   fine

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0d000,d2      ; linea da aspettare = $D0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $D0
        BNE.S   Waity1

;           ____            .    _  .
;          /# /_\_          |\_|/__/|
;         |  |/o\o\        / / \/ \  \
;         |  \\_/_/      /__|O||O|__ \
;        / |_   |       |/_ \_/\_/ _\ |
;       |  ||\_ ~|      | | (____) | ||
;       |  ||| \/       \/\___/\__/  //
;       |  |||_         (_/         ||
;        \//  |          |          ||
;         ||  |          |          ||\
;         ||_  \          \        //_/
;         \_|  o|          \______//
;         /\___/         __ || __||
;        /  ||||__      (____(____)
;           (___)_)
```

```
        bsr.s   CancellaOggetto         ; cancella il bob

        move    #$0b0,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito la cancellazione

        bsr.s   DisegnaOggetto          ; disegna il bob

        move    #$b00,$180(a5)          ; schermo verde scuro quando il
                                        ; PROCESSORE ha finito il disegno

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit_coppermonitor

        move.w  #$000,$180(a5)

        bra.s   mouse

fine:
        rts


;***************************************************************************
; Questa routine cancella il BOB mediante il blitter. La cancellazione
; viene fatta sul rettangolo che racchiude il bob
;***************************************************************************

CancellaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.w  #$0f0,$180(a5)          ; coppermonitor! schermo verde durante
                                        ; la cancellazione.

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0004,$66(a5)          ; BLTDMOD=40-36=4
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; cancella il rettangolo che racchiude
```

```
                                ; il BOB

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop2

        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;***************************************************************************

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche’ vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$09f0,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.w  #$f00,$180(a5)

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l’ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=0
                                        ; BLTDMOD=40-36=4 come al solito
        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)

        lea     2*18*45(a1),a1          ; punta al prossimo plane sorgente
                                        ; ogni plane e‘ largo 18 words e alto
                                        ; 45 righe
```

```
        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    0       ; qui viene memorizzata la X dell'oggetto
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse

;******************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.raw

;******************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end
```

```
;*****************************************************************************

In questo esempio mostriamo l'effetto del bit Blitter Nasty.
Come abbiamo spiegato questo bit ha effetto solo se il codice risiede
in memoria chip e la cache istruzioni del processore e' disabilitata.
Per caricare il codice in CHIP RAM abbiamo specificato una SECTION CODE_C
Per disabilitare la cache (se l'avete) potete usare il comando CPU
del sistema operativo (oppure andarvi a leggere la lezione sui processori
680x0 !!). A questo punto (praticamente ora siamo nelle condizioni di
un Amiga 500), potete eseguire il programma settando a 0 oppure a 1
il bit 10 di DMACON. Per farlo basta che variate la label DMASET che
si trova all'inizio del listato. Se ponete:

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
                                        ; Blitter Nasty OFF

Avete il Blitter Nasty (bit 10) a 0, e il processore ogni tanto ha
la precedenza.
Se ponete:

                ;5432109876543210
DMASET  EQU     %1000011111000000       ; copper,bitplane,blitter DMA
                                        ; Blitter Nasty ON

Avete il Blitter Nasty (bit 10) a 1, e la precedenza e' sempre del blitter.
Potete verificare che nel secondo caso la blittata e' piu' veloce.
```

Lezione10e6

```
; Lezione10e6.s Versione ottimizzata di lezione10c4.s (Effetto riflettore)
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #5-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
```

```
; qui il blitter e' sicuramente fermo perche' ha provveduto la startup
; quindi possiamo tranquillamente settare i registri.
; I seguenti registri sono usati sempre con gli stessi valori, quindi
; li inizializziamo una volta per tutte all'inizio del programma.

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM/BLTALWM
        move.w  #$0000,$42(a5)          ; BLTCON1 modo ascendente
        move.l  #$00200000,$62(a5)      ; BLTBMOD (40-8=32=$20)
                                        ; BLTAMOD (=0)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse2:

Loop:
        cmp.b   #$ff,$6(a5)     ; VHPOSR - aspetta la linea $ff
        bne.s   loop
Aspetta:
        cmp.b   #$ff,$6(a5)     ; ancora linea $ff?
        beq.s   Aspetta

        bsr.s   ClearScreen     ; pulisci schermo
        bsr.w   SpostaMaschera  ; sposta posizione riflettore
        bsr.s   Riflettore      ; routine effetto

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts

;***************************************************************************
; Questa routine cancella la porzione di schermo interessata dalla blittata
;***************************************************************************

ClearScreen:
        moveq   #5-1,d7                 ; 5 bit-planes
        lea     BITPLANE1+100*40,a1     ; indirizzo zona da cancellare (plane1)

        move.w  #(64*39)+20,d5          ; valore da scrivere in BLTSIZE
                                        ; lo mettiamo in D5 per ottimizzare
                                        ; la scrittura

        btst    #6,2(a5)                ; dmaconr
WBlit1a:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit1a                 ; prima di modificare i registri

        move.w  #$0100,$40(a5)          ; BLTCON0. Cancellazione
        move.w  #$0000,$66(a5)          ; BLTDMOD questi 2 registri sono usati
                                        ; con valori diversi nella routine
                                        ; Riflettore, quindi devono essere
                                        ; reinizializzati ogni volta
                                        ; e' comunque necessario farlo una
                                        ; volta sola, fuori dal loop.

ClearLoop:
```

```
        btst    #6,2(a5)                ; dmaconr
WBlit1b:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit1b                 ; prima di blittare

        move.l  a1,$54(a5)
        move.w  d5,$58(a5)              ; scrivi BLTSIZE
                                        ; il valore e' stato precedentemente
                                        ; scritto in D5

        add.l   #256*40,a1              ; indirizzo prossimo plane
        dbra    d7,Clearloop
        rts

;***************************************************************************
; Questa routine realizza l'effetto riflettore. Viene effettuata un'operazione
; di AND tra la figura e una maschera
;***************************************************************************

;          |\_._/|        |,\__/|        |\__/,|
;          | o o |        |  o o|        |o o  |
;          (  T  )        (   T )        ( T   )
;         .^'-^-'^.      .^'--^'^.      .^'^--'^.
;         '.  ;  .'      '.  ;  .'      '.  ;  .'
;         | | | | |      | | | | |      | | | | |
;        ((_(((|)))_))    ((_(((|)))_))    ((_(((|)))_))

Riflettore:
        moveq   #5-1,d7                 ; 5 bit-planes
        lea     Figura+40,a0            ; ind. figura
        lea     BITPLANE1+100*40,a1     ; ind. destinazione

        move.w  MascheraX(PC),d0 ; posizione riflettore
        move.w  d0,d2           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$0dc0,d0       ; ...giusti per inserirsi nel registro BLTCON0
                                ; notate LF=$C0 (cioe' AND tra A e B)
        lsr.w   #3,d2           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d2       ; escludo il bit 0 del
        add.w   d2,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto nella figura
        add.w   d2,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        move.l  #Maschera,a2            ; valore da scrivere in BLTAPT
                                        ; (puntatore maschera)
                                        ; lo mettiamo in A2 per ottimizzare
                                        ; la scrittura

        move.w  #(64*39)+4,d5           ; valore da scrivere in BLTSIZE
                                        ; lo mettiamo in D5 per ottimizzare
                                        ; la scrittura

        btst    #6,2(a5)                ; dmaconr
```

```
WBlit2a:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2a                 ; prima di modificare i registri

        move.w  #32,$66(a5)             ; BLTDMOD (40-8=32)
        move.w  d0,$40(a5)              ; BLTCON0 questi 2 registri sono usati
                                        ; con valori diversi nella routine
                                        ; ClearScreen, quindi devono essere
                                        ; reinizializzati ogni volta
                                        ; e‘ comunque necessario farlo una
                                        ; volta sola, fuori dal loop.

Drawloop:
        btst    #6,2(a5)                ; dmaconr
WBlit2b:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2b                 ; prima di blittare

        move.l  a2,$50(a5)              ; BLTAPT  puntatore maschera
                                        ; il valore e‘ stato precedentemente
                                        ; scritto in A2
        move.l  a0,$4c(a5)              ; BLTBPT  puntatore figura
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  d5,$58(a5)              ; scrivi BLTSIZE
                                        ; il valore e‘ stato precedentemente
                                        ; scritto in D5

        add.w   #56*40,a0               ; ind. prossimo plane figura
        add.w   #256*40,a1              ; ind. prossimo plane destinazione
        dbra    d7,Drawloop
        rts

;*****************************************************************************
; Questa routine legge la coordinata orizzontale da una tabella
; e la memorizza nella variabile MASCHERAX
;*****************************************************************************

SpostaMaschera:
        ADDQ.L  #2,TABXPOINT            ; Fai puntare alla word successiva
        MOVE.L  TABXPOINT(PC),A0        ; indirizzo contenuto in long TABXPOINT
                                        ; copiato in a0
        CMP.L   #FINETABX-2,A0          ; Siamo all’ultima word della TAB?
        BNE.S   NOBSTARTX               ; non ancora? allora continua
        MOVE.L  #TABX-2,TABXPOINT       ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        MOVE.W  (A0),MascheraX          ; copia il valore nella variabile
        rts

MascheraX:
                dc.w    0       ; posizione attuale maschera
TABXPOINT:
                dc.l    TABX    ; puntatore alla tabella

; tabella posizioni maschera

TABX:
        DC.W    $12,$16,$19,$1D,$21,$25,$28,$2C,$30,$34
        DC.W    $37,$3B,$3F,$43,$46,$4A,$4E,$51,$55,$58
        DC.W    $5C,$60,$63,$67,$6A,$6E,$71,$74,$78,$7B
        DC.W    $7F,$82,$85,$89,$8C,$8F,$92,$95,$98,$9C
        DC.W    $9F,$A2,$A5,$A8,$AA,$AD,$B0,$B3,$B6,$B8
        DC.W    $BB,$BE,$C0,$C3,$C5,$C8,$CA,$CC,$CF,$D1
```

```
        DC.W    $D3,$D5,$D8,$DA,$DC,$DE,$E0,$E1,$E3,$E5
        DC.W    $E7,$E8,$EA,$EC,$ED,$EE,$F0,$F1,$F2,$F4
        DC.W    $F5,$F6,$F7,$F8,$F9,$FA,$FB,$FB,$FC,$FD
        DC.W    $FD,$FE,$FE,$FF,$FF,$FF,$100,$100,$100,$100
        DC.W    $100,$100,$100,$100,$FF,$FF,$FF,$FE,$FE,$FD
        DC.W    $FD,$FC,$FB,$FB,$FA,$F9,$F8,$F7,$F6,$F5
        DC.W    $F4,$F2,$F1,$F0,$EE,$ED,$EC,$EA,$E8,$E7
        DC.W    $E5,$E3,$E1,$E0,$DE,$DC,$DA,$D8,$D5,$D3
        DC.W    $D1,$CF,$CC,$CA,$C8,$C5,$C3,$C0,$BE,$BB
        DC.W    $B8,$B6,$B3,$B0,$AD,$AA,$A8,$A5,$A2,$9F
        DC.W    $9C,$98,$95,$92,$8F,$8C,$89,$85,$82,$7F
        DC.W    $7B,$78,$74,$71,$6E,$6A,$67,$63,$60,$5C
        DC.W    $58,$55,$51,$4E,$4A,$46,$43,$3F,$3B,$37
        DC.W    $34,$30,$2C,$28,$25,$21,$1D,$19,$16,$12
FINETABX:

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$5200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000
        dc.w $f0,$0000,$f2,$0000

Colours:
        dc.w    $180,0,$182,$f10,$184,$f21,$186,$f42
        dc.w    $188,$f53,$18a,$f63,$18c,$f74,$18e,$f85
        dc.w    $190,$f96,$192,$fa6,$194,$fb7,$196,$fb8
        dc.w    $198,$fc9,$19a,$f21,$19c,$f10,$19e,$f00
        dc.w    $1a0,$eff,$1a2,$eff,$1a4,$dff,$1a6,$dff
        dc.w    $1a8,$cff,$1aa,$bef,$1ac,$bef,$1ae,$adf
        dc.w    $1b0,$9df,$1b2,$9cf,$1b4,$8bf,$1b6,$7bf
        dc.w    $1b8,$7af,$1ba,$69f,$1bc,$68f,$1be,$57f

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; qui c'e' il disegno, largo 320 pixel, alto 56 linee e formato da 5 plane

Figura:
        incbin  lava320*56*5.raw

;***************************************************************************

; Questa e' la maschera. E' una figura formata da un solo bitplane,
; alta 39 linee e larga 4 words
```

```
Maschera:
        dc.l    $00007fc0,$00000000,$0003fff8,$00000000,$000ffffe,$00000000
        dc.l    $001fffff,$00000000,$007fffff,$c0000000,$00ffffff,$e0000000
        dc.l    $01ffffff,$f0000000,$03ffffff,$f8000000,$03ffffff,$f8000000
        dc.l    $07ffffff,$fc000000,$0fffffff,$fe000000,$0fffffff,$fe000000
        dc.l    $1fffffff,$ff000000,$1fffffff,$ff000000,$1fffffff,$ff000000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $3fffffff,$ff800000,$3fffffff,$ff800000,$3fffffff,$ff800000
        dc.l    $1fffffff,$ff000000,$1fffffff,$ff000000,$1fffffff,$ff000000
        dc.l    $0fffffff,$fe000000,$0fffffff,$fe000000,$07ffffff,$fc000000
        dc.l    $03ffffff,$f8000000,$03ffffff,$f8000000,$01ffffff,$f0000000
        dc.l    $00ffffff,$e0000000,$007fffff,$c0000000,$001fffff,$00000000
        dc.l    $000ffffe,$00000000,$0003fff8,$00000000,$00007fc0,$00000000

;***************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256
BITPLANE4:
        ds.b    40*256
BITPLANE5:
        ds.b    40*256

        end

;***************************************************************************

Questo esempio e‘ la versione ottimizzata di lezione10c4.s. le ottimizzazioni
sono spiegate nel listato.
```

26.5 Lezione10f1

```
; Lezione10f1.s Tanti BOB con sfondo "finto". C’e’ un "errore" da sistemare!
;               Tasto sinistro per uscire.

        SECTION bau,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


; costanti bordi.

Lowest_Floor    equ     200     ; bordo in basso
Right_Side      equ     287     ; bordo a destra
```

```
START:

; puntiamo i bitplanes
        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

;       Puntiamo il quarto bitplane (lo sfondo)

        LEA     BPLPOINTERS,A0          ; puntatori COP
        move.l  #SfondoFinto,d0         ; indirizzo sfondo
        move.w  d0,30(a0)               ; lo sfondo e‘ il bitplane 4
        swap    d0
        move.w  d0,26(a0)               ; scrivi word alta

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   CancellaSchermo         ; cancella lo schermo

        lea     Oggetto_1,a4            ; indirizzo primo oggetto
        moveq   #6-1,d6                 ; 6 oggetti

Ogg_loop:
        bsr.s   MuoviOggetto            ; muove il bob
        bsr.w   DisegnaOggetto          ; disegna il bob

        addq.l  #8,a4                   ; punta al prossimo oggetto

        dbra    d6,Ogg_loop

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)
        bne.s   WBlit_coppermonitor

        move.w  #$aaa,$180(a5)          ; copper monitor: colore grigio

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:
```

```
        rts


;*****************************************************************************
; Questa routine muove un bob controllando che non superi i bordi
; A4 - punta alla struttura dati che contiene la posizione e la velocita'
;      del bob
;*****************************************************************************

MuoviOggetto:
        move.w  (a4),d0                 ; posizione X
        move.w  2(a4),d1                ; posizione Y
        move.w  4(a4),d2                ; dx (velocita' X)
        move.w  6(a4),d3                ; dy (velocita' Y)
        add.w   d2,d0                   ; x = x + dx
        add.w   d3,d1                   ; y = y + dy

        btst    #15,d1                  ; controlla bordo alto (Y=0)
        beq.s   UO_NoBounce4            ; se la Y e' negativa...
        neg.w   d1                      ; .. fa il rimbalzo
        neg.w   d3                      ; inverti direzione del moto
UO_NoBounce4:

        cmp.w   #Lowest_Floor,d1        ; controlla bordo in basso
        blt.s   UO_NoBounce1

        neg.w   d3                      ; cambia il segno della velocita' dy
                                        ; invertendo la direzione del moto
        move.w  #Lowest_Floor,d1        ; riparti dal bordo
UO_NoBounce1:

        cmp.w   #Right_Side,d0          ; controlla bordo destro
        blt.s   UO_NoBounce2            ; se supera il bordo destro..
        sub.w   #Right_Side,d0          ; distanza dal bordo
        neg.w   d0                      ; inverti la distanza
        add.w   #Right_Side,d0          ; aggiungi coordinata bordo
        neg.w   d2                      ; inverti direzione del moto
UO_NoBounce2:
        btst    #15,d0                  ; controlla bordo sinistro (X=0)
        beq.s   UO_NoBounce3            ; se la X e' negativa...
        neg.w   d0                      ; .. fa il rimbalzo
        neg.w   d2                      ; inverti direzione del moto
UO_NoBounce3:
        move.w  d0,(a4)                 ; aggiorna posizione e velocita'
        move.w  d1,2(a4)
        move.w  d2,4(a4)
        move.w  d3,6(a4)

        rts


;*****************************************************************************
; Questa routine disegna un BOB.
; A4 - punta alla struttura dati che contiene la posizione e la velocita'
;      del bob
;*****************************************************************************

;               |\__/,|   (`\
;             _.|o o  |_   ) )
;         ---(((---(((---------
```

```
DisegnaOggetto:
        lea     BITPLANE1,a0    ; indirizzo bitplane
        move.w  2(a4),d0        ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga occupa 40 bytes

        add.l   d0,a0           ; aggiungi offset Y

        move.w  (a4),d0         ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$0FCA,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.l   #$0000fffe,d1   ; escludo il bit 0 del
        add.l   d1,a0           ; aggiungi l'offset X, trovando l'indirizzo
                                ; della destinazione

        lea     Ball_Bob,a1             ; puntatore alla figura
        lea     Ball_Mask,a2            ; puntatore alla maschera
        moveq   #3-1,d7                 ; bitplane counter

DrawLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)
        bne.s   WBlit2

        move.w  d0,$40(a5)              ; BLTCON0 - scrivi valore di shift
        move.w  d0,d1                   ; copia valore di BLTCON0,
        and.w   #$f000,d1               ; seleziona valore di shift..
        move.w  d1,$42(a5)              ; e scrivilo in BLTCON1 (per canale B)

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM e BLTLWM

        move.w  #$FFFE,$64(a5)          ; BLTAMOD
        move.w  #$FFFE,$62(a5)          ; BLTBMOD

        move.w  #40-6,$66(a5)           ; BLTDMOD
        move.w  #40-6,$60(a5)           ; BLTCMOD

        move.l  a2,$50(a5)              ; BLTAPT - puntatore maschera
        move.l  a1,$4c(a5)              ; BLTBPT - puntatore figura
        move.l  a0,$48(a5)              ; BLTCPT - puntatore sfondo
        move.l  a0,$54(a5)              ; BLTDPT - puntatore bitplanes

        move.w  #(31*64)+3,$58(a5)      ; BLTSIZE - altezza 31 linee
                                        ; largh. 3 word (48 pixel).

        add.l   #4*31,a1                ; indirizzo prossimo plane immagine
        add.l   #40*256,a0              ; indirizzo prossimo plane destinazione
        dbra    d7,DrawLoop



        rts
```

```
;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
;***************************************************************************

CancellaSchermo:
        moveq   #3-1,d7                 ; 3 bitplanes
        lea     BITPLANE1,a0            ; indirizzo schermo

canc_loop:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                 ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move    #$0000,$66(a5)          ; BLTDMOD=0
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo

        add.l   #40*256,a0              ; indirizzo prossimo plane destinazione
        dbra    d7,canc_loop
        rts


; dati oggetti
; queste sono le strutture dati che contengono velocita‘ e posizione dei bobs.
; ogni struttra dati si compone di 4 words che contengono nell’ordine:
; POSIZIONE X, POSIZIONE Y, VELOCITA‘ X, VELOCITA‘ Y

Oggetto_1:
        dc.w    32,53           ;  x / y   - posizione
        dc.w    -3,1            ; dx / dy  - velocita‘

Oggetto_2:
        dc.w    132,62          ;  x / y   - posizione
        dc.w    2,-1            ; dx / dy  - velocita‘

Oggetto_3:
        dc.w    232,42          ;  x / y   - posizione
        dc.w    3,1             ; dx / dy  - velocita‘

Oggetto_4:
        dc.w    2,20            ;  x / y   - posizione
        dc.w    -5,1            ; dx / dy  - velocita‘

Oggetto_5:
        dc.w    60,80           ;  x / y   - posizione
        dc.w    6,1             ; dx / dy  - velocita‘

Oggetto_6:
        dc.w    50,75           ;  x / y   - posizione
        dc.w    -5,1            ; dx / dy  - velocita‘

;***************************************************************************

        SECTION MY_COPPER,CODE_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
```

```
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

        dc.w    $108,0          ; MODULO
        dc.w    $10a,0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000

        dc.w    $180,$000       ; color0 - sfondo
        dc.w    $190,$000

        dc.w    $182,$0A0                       ; colori da 1 a 7
        dc.w    $184,$040
        dc.w    $186,$050
        dc.w    $188,$061
        dc.w    $18A,$081
        dc.w    $18C,$020
        dc.w    $18E,$6F8

        dc.w    $192,$0A0               ; colori da 9 a 15
        dc.w    $194,$040               ; sono gli stessi valori
        dc.w    $196,$050               ; caricati nei registri da 1 a 7
        dc.w    $198,$061
        dc.w    $19a,$081
        dc.w    $19c,$020
        dc.w    $19e,$6F8

        dc.w    $190,$345       ; colore 8 - pixel ad 1 dello sfondo

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $8007,$fffe     ; aspetta riga $80
        dc.w    $100,$4200      ; bplcon0 - 4 bitplanes lowres
                                ; attiva il bitplane 4 (sfondo)

; in questo spazio e' visualizzata la parte dello sfondo

        dc.w    $e007,$fffe     ; aspetta riga $e0
        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Figura Bob
Ball_Bob:
 DC.W $0000,$0000,$0000,$0000,$0000,$0000,$003F,$8000   ; plane 1
 DC.W $00C1,$E000,$017C,$E000,$02FE,$3000,$05FF,$5400
 DC.W $07FF,$1800,$0BFE,$AC00,$03FF,$1A00,$0BFE,$AC00
 DC.W $11FF,$1A00,$197D,$2C00,$0EAA,$1A00,$1454,$DC00
 DC.W $0E81,$3800,$0154,$F400,$02EB,$F000,$015F,$D000
 DC.W $00B5,$A000,$002A,$8000,$0000,$0000,$0000,$0000
 DC.W $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
 DC.W $0000,$0000,$0000,$0000,$0000,$0000
```

```
 DC.W $000F,$E000,$007F,$FC00,$01FF,$FF00,$03FF,$FF80   ; plane 2
 DC.W $07C1,$FFC0,$0F00,$FFE0,$1E00,$3FF0,$3C40,$5FF8
 DC.W $3CE0,$1FF8,$7840,$2FFC,$7800,$1FFC,$7800,$2FFC
 DC.W $F800,$1FFE,$F800,$2FFE,$FE00,$1FFE,$FC00,$DFFE
 DC.W $FE81,$3FFE,$FF54,$FFFE,$FFEB,$FFFE,$7FFF,$FFFC
 DC.W $7FFF,$FFFC,$7FFF,$FFFC,$3FFF,$FFF8,$3FFF,$FFF8
 DC.W $1FFF,$FFF0,$0FFF,$FFE0,$07FF,$FFC0,$03FF,$FF80
 DC.W $01FF,$FF00,$007F,$FC00,$000F,$E000

 DC.W $000F,$E000,$007F,$FC00,$01E0,$7F00,$0380,$0F80   ; plane 3
 DC.W $073E,$0AC0,$0CFF,$0560,$198F,$C2F0,$3347,$A0B8
 DC.W $32EB,$E158,$6647,$D0AC,$660B,$E05C,$4757,$D0AC
 DC.W $C7AF,$E05E,$A7FF,$D02E,$C1FF,$E05E,$A3FF,$202E
 DC.W $D17E,$C05E,$E0AB,$002E,$D014,$005E,$6800,$00AC
 DC.W $7000,$02DC,$7400,$057C,$2800,$0AF8,$3680,$55F8
 DC.W $1D54,$AAF0,$0EAB,$55E0,$0754,$ABC0,$03EB,$FF80
 DC.W $01FE,$FF00,$007F,$FC00,$000F,$E000

; Maschera Bob
Ball_MASK:
 DC.W $000F,$E000,$007F,$FC00,$01FF,$FF00,$03FF,$FF80
 DC.W $07FF,$FFC0,$0FFF,$FFE0,$1FFF,$FFF0,$3FFF,$FFF8
 DC.W $3FFF,$FFF8,$7FFF,$FFFC,$7FFF,$FFFC,$7FFF,$FFFC
 DC.W $FFFF,$FFFE,$FFFF,$FFFE,$FFFF,$FFFE,$FFFF,$FFFE
 DC.W $FFFF,$FFFE,$FFFF,$FFFE,$FFFF,$FFFE,$7FFF,$FFFC
 DC.W $7FFF,$FFFC,$7FFF,$FFFC,$3FFF,$FFF8,$3FFF,$FFF8
 DC.W $1FFF,$FFF0,$0FFF,$FFE0,$07FF,$FFC0,$03FF,$FF80
 DC.W $01FF,$FF00,$007F,$FC00,$000F,$E000


;***************************************************************************

; Sfondo 320 * 100 1 Bitplane, raw normale.

SfondoFinto:
        incbin  "sfondo320*100.raw"

;***************************************************************************

        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256

        end

;***************************************************************************
```

In questo esempio vediamo 6 bob che si muovono su uno sfondo.
Utilizziamo il trucco dello sfondo finto. Siccome pero‘ i 6 bobs sono disegnati
tutti negli stessi planes, dobbiamo comunque blittarli usando la tecnica
del bitplane maschera e del "cookie cut", altrimenti non si sovrapporrebbero
correttamente. La tecnica dello sfondo finto ci consente tuttavia di evitare
il salvataggio ed il ripristino dello sfondo, visto che lo sfondo e‘ formato
da soli zeri. Per cancellare i bobs disegnati nelle vecchie posizioni e‘
suffincente quindi cancellare i planes dedicati ai bobs prima di iniziare
a ridisegnarli.
Per muovere e disegnare i bobs usiamo delle routines parametriche che sono
in grado di gestire tutti i bobs che vogliamo. Invece di passare i parametri

```
attraverso i registri della CPU, usiamo delle "strutture dati", ovverosia
raccogliamo in indirizzi contigui i dati di velocita‘ e posizione di ogni bob,
seguendo sempre lo stesso ordine. Alle routine viene "passato" attraverso il
registro A4 l’indirizzo della struttura dati. In questo modo le routine
sanno che i dati del bob si trovano all’indirizzo puntato da A4 e agli
indirizzi successivi.
Questo programma come potete vedere eseguendolo, non disegna correttamente
i bobs. Per la spiegazione del problema e per la soluzione consultate la
lezione.
```

Lezione10f2

```
; Lezione10f2.s Tanti BOB con sfondo "finto" e "double buffering"
;               Tasto sinistro per uscire.

        SECTION bau,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


; costanti bordi.

Lowest_Floor    equ     200     ; bordo in basso
Right_Side      equ     287     ; bordo a destra


START:
; i primi 3 planes vengono puntati dalla routine ScambiaBuffer

;       Puntiamo il quarto bitplane (lo sfondo)

        LEA     BPLPOINTERS,A0          ; puntatori COP
        move.l  #SfondoFinto,d0         ; indirizzo sfondo
        move.w  d0,30(a0)               ; lo sfondo e‘ il bitplane 4
        swap    d0
        move.w  d0,26(a0)               ; scrivi word alta

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l’AGA
        move.w  #$c00,$106(a5)          ; Disattiva l’AGA
        move.w  #$11,$10c(a5)           ; Disattiva l’AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1
```

```
        bsr.w   ScambiaBuffer           ; questa routine scambia i 2 buffer

        bsr.w   CancellaSchermo         ; cancella lo schermo

        lea     Oggetto_1,a4            ; indirizzo primo oggetto
        moveq   #6-1,d6                 ; 6 oggetti

Ogg_loop:
        bsr.s   MuoviOggetto            ; muove il bob
        bsr.w   DisegnaOggetto          ; disegna il bob

        addq.l  #8,a4                   ; punta al prossimo oggetto

        dbra    d6,Ogg_loop

        btst    #6,2(a5)
WBlit_coppermonitor:
        btst    #6,2(a5)
        bne.s   WBlit_coppermonitor

        move.w  #$aaa,$180(a5)          ; copper monitor: colore grigio

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts

;*****************************************************************************
; Questa routine scambia i 2 buffer scambiando gli indirizzi nelle
; variabili VIEW_BUFFER e  DRAW_BUFFER.
; Inoltre aggiorna nella copperlist le istruzioni che caricano i registri
; BPLxPT, in modo che puntino al nuovo buffer da visualizzare.
;*****************************************************************************

;               |\__/,|   (`\
;             _.|o o  |_   ) )
;         ---(((---(((---------

ScambiaBuffer:
        move.l  draw_buffer(pc),d0              ; scambia il contenuto
        move.l  view_buffer(pc),draw_buffer     ; delle variabili
        move.l  d0,view_buffer                  ; in d0 c'e' l'indirizzo
                                                ; del nuovo buffer
                                                ; da visualizzare

; aggiorna la copperlist puntando i bitplanes del nuovo buffer da visualizzare

        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        rts

;*****************************************************************************
```

```
; Questa routine muove un bob controllando che non superi i bordi
; A4 - punta alla struttura dati che contiene la posizione e la velocita'
;      del bob
;*****************************************************************************

MuoviOggetto:
        move.w  (a4),d0                 ; posizione X
        move.w  2(a4),d1                ; posizione Y
        move.w  4(a4),d2                ; dx (velocita' X)
        move.w  6(a4),d3                ; dy (velocita' Y)
        add.w   d2,d0                   ; x = x + dx
        add.w   d3,d1                   ; y = y + dy

        btst    #15,d1                  ; controlla bordo alto (Y=0)
        beq.s   UO_NoBounce4            ; se la Y e' negativa...
        neg.w   d1                      ; .. fa il rimbalzo
        neg.w   d3                      ; inverti direzione del moto
UO_NoBounce4:

        cmp.w   #Lowest_Floor,d1        ; controlla bordo in basso
        blt.s   UO_NoBounce1

        neg.w   d3                      ; cambia il segno della velocita' dy
                                        ; invertendo la direzione del moto
        move.w  #Lowest_Floor,d1        ; riparti dal bordo
UO_NoBounce1:

        cmp.w   #Right_Side,d0          ; controlla bordo destro
        blt.s   UO_NoBounce2            ; se supera il bordo destro..
        sub.w   #Right_Side,d0          ; distanza dal bordo
        neg.w   d0                      ; inverti la distanza
        add.w   #Right_Side,d0          ; aggiungi coordinata bordo
        neg.w   d2                      ; inverti direzione del moto
UO_NoBounce2:
        btst    #15,d0                  ; controlla bordo sinistro (X=0)
        beq.s   UO_NoBounce3            ; se la X e' negativa...
        neg.w   d0                      ; .. fa il rimbalzo
        neg.w   d2                      ; inverti direzione del moto
UO_NoBounce3:
        move.w  d0,(a4)                 ; aggiorna posizione e velocita'
        move.w  d1,2(a4)
        move.w  d2,4(a4)
        move.w  d3,6(a4)

        rts


;*****************************************************************************
; Questa routine disegna un BOB.
; A4 - punta alla struttura dati che contiene la posizione e la velocita'
;      del bob
;*****************************************************************************

DisegnaOggetto:
        move.l  draw_buffer(pc),a0      ; indirizzo buffer disegno
        move.w  2(a4),d0        ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga occupa 40 bytes

        add.l   d0,a0           ; aggiungi offset Y

        move.w  (a4),d0         ; coordinata X
        move.w  d0,d1           ; copia
```

```
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        or.w    #$0FCA,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.l   #$0000fffe,d1   ; escludo il bit 0 del
        add.l   d1,a0           ; aggiungi l'offset X, trovando l'indirizzo
                                ; della destinazione

        lea     Ball_Bob,a1             ; puntatore alla figura
        lea     Ball_Mask,a2            ; puntatore alla maschera
        moveq   #3-1,d7                 ; bitplane counter

DrawLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)
        bne.s   WBlit2

        move.w  d0,$40(a5)              ; BLTCON0 - scrivi valore di shift
        move.w  d0,d1                   ; copia valore di BLTCON0,
        and.w   #$f000,d1               ; seleziona valore di shift..
        move.w  d1,$42(a5)              ; e scrivilo in BLTCON1 (per canale B)

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM e BLTLWM

        move.w  #$FFFE,$64(a5)          ; BLTAMOD
        move.w  #$FFFE,$62(a5)          ; BLTBMOD

        move.w  #40-6,$66(a5)           ; BLTDMOD
        move.w  #40-6,$60(a5)           ; BLTCMOD

        move.l  a2,$50(a5)              ; BLTAPT - puntatore maschera
        move.l  a1,$4c(a5)              ; BLTBPT - puntatore figura
        move.l  a0,$48(a5)              ; BLTCPT - puntatore sfondo
        move.l  a0,$54(a5)              ; BLTDPT - puntatore bitplanes

        move.w  #(31*64)+3,$58(a5)      ; BLTSIZE - altezza 31 linee
                                        ; largh. 3 word (48 pixel).

        add.l   #4*31,a1                ; indirizzo prossimo plane immagine
        add.l   #40*256,a0              ; indirizzo prossimo plane destinazione
        dbra    d7,DrawLoop



        rts


;******************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
;******************************************************************************

CancellaSchermo:
        moveq   #3-1,d7                 ; 3 bitplanes
        move.l  draw_buffer(pc),a0      ; indirizzo buffer disegno
```

```
canc_loop:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move    #$0000,$66(a5)          ; BLTDMOD=0
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo

        add.l   #40*256,a0              ; indirizzo prossimo plane destinazione
        dbra    d7,canc_loop
        rts

; puntatori ai 2 buffer
view_buffer     dc.l    BITPLANE1       ; buffer visualizzato
draw_buffer     dc.l    BITPLANE1b      ; buffer di disegno


; dati oggetti
; queste sono le strutture dati che contengono velocita‘ e posizione dei bobs.
; ogni struttra dati si compone di 4 words che contengono nell’ordine:
; POSIZIONE X, POSIZIONE Y, VELOCITA‘ X, VELOCITA‘ Y

Oggetto_1:
        dc.w    32,53           ;  x / y   - posizione
        dc.w    -3,1            ; dx / dy  - velocita‘

Oggetto_2:
        dc.w    132,65          ;  x / y   - posizione
        dc.w    2,-1            ; dx / dy  - velocita‘

Oggetto_3:
        dc.w    232,42          ;  x / y   - posizione
        dc.w    3,1             ; dx / dy  - velocita‘

Oggetto_4:
        dc.w    2,20            ;  x / y   - posizione
        dc.w    -5,1            ; dx / dy  - velocita‘

Oggetto_5:
        dc.w    60,80           ;  x / y   - posizione
        dc.w    6,1             ; dx / dy  - velocita‘

Oggetto_6:
        dc.w    50,75           ;  x / y   - posizione
        dc.w    -5,1            ; dx / dy  - velocita‘

;******************************************************************************

        SECTION MY_COPPER,CODE_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
```

```
        dc.w    $108,0          ; MODULO
        dc.w    $10a,0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000
        dc.w $ec,$0000,$ee,$0000

        dc.w    $180,$000       ; color0 - sfondo
        dc.w    $190,$000

        dc.w    $182,$0A0                       ; colori da 1 a 7
        dc.w    $184,$040
        dc.w    $186,$050
        dc.w    $188,$061
        dc.w    $18A,$081
        dc.w    $18C,$020
        dc.w    $18E,$6F8

        dc.w    $192,$0A0                       ; colori da 9 a 15
        dc.w    $194,$040                       ; sono gli stessi valori
        dc.w    $196,$050                       ; caricati nei registri da 1 a 7
        dc.w    $198,$061
        dc.w    $19a,$081
        dc.w    $19c,$020
        dc.w    $19e,$6F8

        dc.w    $190,$345       ; colore 8 - pixel ad 1 dello sfondo

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $8007,$fffe     ; aspetta riga $80
        dc.w    $100,$4200      ; bplcon0 - 4 bitplanes lowres
                                ; attiva il bitplane 4 (sfondo)

; in questo spazio e' visualizzata la parte dello sfondo

        dc.w    $e007,$fffe     ; aspetta riga $e0
        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Figura Bob
Ball_Bob:
 DC.W $0000,$0000,$0000,$0000,$0000,$0000,$003F,$8000   ; plane 1
 DC.W $00C1,$E000,$017C,$E000,$02FE,$3000,$05FF,$5400
 DC.W $07FF,$1800,$0BFE,$AC00,$03FF,$1A00,$0BFE,$AC00
 DC.W $11FF,$1A00,$197D,$2C00,$0EAA,$1A00,$1454,$DC00
 DC.W $0E81,$3800,$0154,$F400,$02EB,$F000,$015F,$D000
 DC.W $00B5,$A000,$002A,$8000,$0000,$0000,$0000,$0000
 DC.W $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
 DC.W $0000,$0000,$0000,$0000,$0000,$0000

 DC.W $000F,$E000,$007F,$FC00,$01FF,$FF00,$03FF,$FF80   ; plane 2
 DC.W $07C1,$FFC0,$0F00,$FFE0,$1E00,$3FF0,$3C40,$5FF8
 DC.W $3CE0,$1FF8,$7840,$2FFC,$7800,$1FFC,$7800,$2FFC
 DC.W $F800,$1FFE,$F800,$2FFE,$FE00,$1FFE,$FC00,$DFFE
 DC.W $FE81,$3FFE,$FF54,$FFFE,$FFEB,$FFFE,$7FFF,$FFFC
```

```
 DC.W $7FFF,$FFFC,$7FFF,$FFFC,$3FFF,$FFF8,$3FFF,$FFF8
 DC.W $1FFF,$FFF0,$0FFF,$FFE0,$07FF,$FFC0,$03FF,$FF80
 DC.W $01FF,$FF00,$007F,$FC00,$000F,$E000

 DC.W $000F,$E000,$007F,$FC00,$01E0,$7F00,$0380,$0F80   ; plane 3
 DC.W $073E,$0AC0,$0CFF,$0560,$198F,$C2F0,$3347,$A0B8
 DC.W $32EB,$E158,$6647,$D0AC,$660B,$E05C,$4757,$D0AC
 DC.W $C7AF,$E05E,$A7FF,$D02E,$C1FF,$E05E,$A3FF,$202E
 DC.W $D17E,$C05E,$E0AB,$002E,$D014,$005E,$6800,$00AC
 DC.W $7000,$02DC,$7400,$057C,$2800,$0AF8,$3680,$55F8
 DC.W $1D54,$AAF0,$0EAB,$55E0,$0754,$ABC0,$03EB,$FF80
 DC.W $01FE,$FF00,$007F,$FC00,$000F,$E000

; Maschera Bob
Ball_MASK:
 DC.W $000F,$E000,$007F,$FC00,$01FF,$FF00,$03FF,$FF80
 DC.W $07FF,$FFC0,$0FFF,$FFE0,$1FFF,$FFF0,$3FFF,$FFF8
 DC.W $3FFF,$FFF8,$7FFF,$FFFC,$7FFF,$FFFC,$7FFF,$FFFC
 DC.W $FFFF,$FFFE,$FFFF,$FFFE,$FFFF,$FFFE,$FFFF,$FFFE
 DC.W $FFFF,$FFFE,$FFFF,$FFFE,$FFFF,$FFFE,$7FFF,$FFFC
 DC.W $7FFF,$FFFC,$7FFF,$FFFC,$3FFF,$FFF8,$3FFF,$FFF8
 DC.W $1FFF,$FFF0,$0FFF,$FFE0,$07FF,$FFC0,$03FF,$FF80
 DC.W $01FF,$FF00,$007F,$FC00,$000F,$E000


;***************************************************************************

; Sfondo 320 * 100 1 Bitplane, raw normale.

SfondoFinto:
        incbin  "sfondo320*100.raw"

;***************************************************************************

        SECTION bitplane,BSS_C
; Questi sono i bitplanes del primo buffer
BITPLANE1:
        ds.b    40*256
BITPLANE2:
        ds.b    40*256
BITPLANE3:
        ds.b    40*256

; Questi sono i bitplanes del secondo buffer
BITPLANE1b:
        ds.b    40*256
BITPLANE2b:
        ds.b    40*256
BITPLANE3b:
        ds.b    40*256

        end

;***************************************************************************
```

In questo esempio risolviamo il problema dell'esempio lezione10f1.s mediante
la tecnica del double buffering. Questa tecnica consiste nell'uso di 2
buffer separati, uno che viene visualizzato e uno sul quale si disegna
che vengono scambiati ad ogni Vertical Blank.
Gli indirizzi dei 2 buffer sono contenuti in 2 variabili. La variabile
draw_buffer contiene l'indirizzo del buffer usato per disegnare nel
fotogramma corrente. Le routine che modificano lo schermo (CancellaSchermo

e DisegnaOggetto) leggono da draw_buffer l'indirizzo del buffer di disegno
e su di esso effettuano le loro operazioni.
La variabile view_buffer contiene l'indirizzo del buffer attualmente
visualizzato.
La routine ScambiaBuffer provvede a scambiare ad ogni fotogramma il contenuto
delle 2 variabili. Inoltre essa fa in modo che venga visualizzato il giusto
buffer (quello il cui indirizzo e' memorizzato in view_buffer). Infatti,
per visualizzare alternativamente i 2 buffer, e' necessario che ad ogni
fotogramma la copper list faccia puntare i registri BPLxPT ai bitplane del
buffer di visualizzazione. Questo significa che la copper list deve essere
modificata ad ogni vertical blank, cosa che viene fatta dalla routine
ScambiaBuffer.

26.6 Lezione10g1

```
; Lezione10g1.s BLITTATA, in cui disegnamo rettangoli a righe sullo schermo
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #1-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

; parametri per routine di disegno

        move.w  #16,d0                  ; X vertice superiore sinistro
        move.w  #10,d1                  ; Y vertice superiore sinistro
        move.w  #48,d2                  ; larghezza
        move.w  #20,d3                  ; altezza
        move.w  #$aaaa,d4               ; "pattern" di disegno
        bsr.s   BlitRett                ; esegui la routine di disegno
```

```
mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

; parametri per routine di disegno

        move.w  #64,d0                  ; X vertice superiore sinistro
        move.w  #70,d1                  ; Y vertice superiore sinistro
        move.w  #32,d2                  ; larghezza
        move.w  #40,d3                  ; altezza
        move.w  #$8FF1,d4               ; "pattern" di disegno
        bsr.s   BlitRett                ; esegui la routine di disegno

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        rts

;******************************************************************************
; Questa routine disegna un rettangolo sullo schermo.
;
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
; D4 - "pattern" con cui disegnare un rettangolo
;******************************************************************************

;                   |\__/,|   (‘\
;                   |o o  |__ _) )
;                 _.( T   )  ‘  /
;        n n._    ((_ ‘^--’ /_<  \
;        <" _ }=- ‘‘ ‘-’((((/  ((((/
;         ‘" "

BlitRett:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

; calcolo indirizzo di partenza del blitter

        lea     bitplane1,a1    ; indirizzo bitplane
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a1           ; aggiungi ad indirizzo
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a1           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

; calcolo modulo blitter

        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        move.w  #40,d5          ; larghezza schermo in bytes
        sub.w   d2,d5           ; modulo=larg. schermo-larg. rettangolo

; calcolo dimensione blittata
```

```
        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d2,d3           ; metti insieme le dimensioni

; carica i registri

        move.l  #$01f00000,$40(a5)      ; BLTCON0 e BLTCON1
                                        ; usa SOLO il canale D
                                        ; LF=$F0 (copia da A a D)
                                        ; modo ascendente

        move.w  d4,$74(a5)              ; BLTADAT
        move.w  d5,$66(a5)              ; BLTDMOD
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$aaa      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        SECTION bitplane,BSS_C

BITPLANE1:
        ds.b    40*256

;***************************************************************************

        end
```

In questo esempio usiamo il blitter per disegnare dei rettangoli con un "pattern", ovvero con un motivo grafico che si ripete.
Utilizziamo una routine parametrica, che traccia un rettangolo conoscendo le coordinate del vertice superiore sinistro e le dimensioni (larghezza e altezza) del rettangolo. Per semplificare la routine la larghezza e la posizione X del vertice sono approssimate a multipli di 16.
Anche il "pattern" viene passato come parametro in un registro. In questo modo con una sola routine possiamo tracciare tutti i "pattern" che vogliamo.

```
Il disegno viene realizzato mediante una blittata che prevede l'attivazione del
solo canale D, ma che copia il contenuto del registro BLTADAT in uscita.
Si deve utilizzare lo stesso valore di LF che si usa nella copia normale da A
a D, ma non si deve attivare il canale A. pertanto il valore da scrivere in
BLTCON0 e' $01F0, ovvero LF=$F0 (copia da A a D) e attivo il solo canale D.
Il valore del "pattern" che verra' copiato in uscita viene scritto nel registro
BLTADAT.
```

Lezione10g2

```
; Lezione10g2.s Esempio di OR tra con un canale abilitato e uno disabilitato
;               Tasto destro per eseguire la blittata, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE1,d0   ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #1-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     Figura,a0
        lea     BITPLANE1+20,a1
        bsr.s   copia           ; esegui copia figura 2

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, non cancellare

        bsr.s   BlitOR          ; esegui l'OR tra le 2 figure

mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
```

```
        bne.s   mouse2          ; se no, torna a mouse2:
        rts


;***************************************************************************
; Questa routine copia la figura sullo schermo.
; Prende come parametri
; A0 - indirizzo sorgente
; A1 - indirizzo destinazione
;***************************************************************************

Copia:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #30,$66(a5)             ; BLTDMOD (40-10=30)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*71)+5,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 5 word
        rts                             ; altezza 71 linee

;***************************************************************************
; Questa routine l'OR tra una figura letta attraverso il canale B
; e il valore costante contenuto in BLTADAT
;***************************************************************************

;         |\__/,|   (`\
;         |_ _  |.--.) )
;         ( T   )     /
;        (((^_(((/(((_>

BlitOR:
        btst    #6,2(a5) ; dmaconr
WBlit2:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$05fc0000,$40(a5)      ; BLTCON0 e BLTCON1
                                        ; usa i canali B e D
                                        ; esegue l'OR tra A e B (LF=$FC)
        move.w  #0,$62(a5)              ; BLTBMOD (=0)
        move.w  #30,$66(a5)             ; BLTDMOD (40-10=30)
        move.w  #$CCCC,$74(a5)          ; valore di OR in BLTADAT

        move.l  #Figura,$4c(a5)         ; BLTBPT  puntatore sorgente
        move.l  #BITPLANE1+100*40+10,$54(a5)    ; BLTDPT  puntatore dest.
        move.w  #(64*71)+5,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 5 word
        rts                             ; altezza 71 linee

;***************************************************************************

        SECTION GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$aaa      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

Figura:
        dc.w    $ffff,$ffff,$ffff,$ffff,$fe00,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,$3800,0,0
        dc.w    0,$0003,$ff80,0,0,0,$001f,$fff0,0,0
        dc.w    0,$01ff,$ffff,0,0,0,$0fff,$ffff,$e000,0
        dc.w    0,$ffff,$ffff,$fe00,0,$0007,$ffff,$ffff,$ffc0,0
        dc.w    $007f,$ffff,$ffff,$fffc,0,$03ff,$ffff,$ffff,$ffff,$8000
        dc.w    $3fff,$ffff,$ffff,$ffff,$f800,$7fff,$ffff,$ffff,$ffff,$fc00
        dc.w    $3fff,$ffff,$ffff,$ffff,$f800,$03ff,$ffff,$ffff,$ffff,$8000
        dc.w    $007f,$ffff,$ffff,$fffc,0,$0007,$ffff,$ffff,$ffc0,0
        dc.w    0,$ffff,$ffff,$fe00,0,0,$0fff,$ffff,$e000,0
        dc.w    0,$01ff,$ffff,0,0,0,$001f,$fff0,0,0
        dc.w    0,$0003,$ff80,0,0,0,0,$3800,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $8000,0,0,0,$0200,$8000,0,0,0,$0200
        dc.w    $ffff,$ffff,$ffff,$ffff,$fe00

;*****************************************************************************
```

```
        SECTION bitplane,BSS_C
BITPLANE1:
        ds.b    40*256

        end

;***************************************************************************

In questo esempio eseguiamo un OR tra una figura letta attraverso il canale B
e un valore costante contenuto nel registro BLTADAT.
Per questo teniamo abilitati i canali B e D, e programmiamo il byte LF
in modo che venga eseguito un OR tra le sorgenti A e B.
```

Lezione10g3

```
; Lezione10g3.s Effetto tendina
;               Tasto destro per vedere la figura, sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1          ; se no, aspetta

        moveq   #16-1,d6        ; ripeti per ogni colonna di pixel

        move.w  #%1000000000000000,d5   ; valore della maschera all'inizio.
                                        ; Fa passare solo il pixel piu'
```

```
                                        ; a sinistra della word.

MostraLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130
        BNE.S   Waity1

        bsr.s   BlitAnd         ; disegna la figura

        asr.w   #1,d5                   ; calcola la maschera per la prossima
                                        ; blittata. Fa passare ogni volta un
                                        ; bit in piu' rispetto alla volta
                                        ; precedente.

        dbra    d6,MostraLoop


mouse2:
        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse2          ; se no, torna a mouse2:

        moveq   #16-1,d6        ; ripeti per ogni colonna di pixel
CancellaLoop:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130
        BNE.S   Waity2

        lsr.w   #1,d5                   ; calcola la maschera per la prossima
                                        ; blittata. Fa passare ogni volta un
                                        ; bit in piu' rispetto alla volta
                                        ; precedente.

        bsr.s   BlitAnd                 ; disegna la figura

        dbra    d6,CancellaLoop

fine:
        rts


;******************************************************************************
; Questa routine esegue un AND tra una figura letta attraverso il canale A
; e un valore costante caricato in BLTBDAT. Il risultato viene disegnato
; sullo schermo.
; D5 - contiene il valore costante (maschera) da caricare in BLTBDAT
;******************************************************************************

;          ____
;         .'_  _`.
;         |/ \/ \|
;         || oo ||
;         ||    ||
;        _|\_/\_/|_
;       (|-.____.-|)
```

```
;        `._ -- _.'
;          |_  _|
;            ``

BlitAnd:
        lea     bitplane+100*40+4,a0    ; puntatore destinazione in a0
        lea     figura,a1               ; puntatore sorgente

        moveq   #3-1,d7                 ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d5,$72(a5)              ; scrive maschera in BLTBDAT
        move.l  #$09C00000,$40(a5)      ; BLTCON0 usa i canali A e D
                                        ;         D=A AND B
                                        ; BLTCON1 (nessun modo speciale)
        move.l  #$00000004,$64(a5)      ; BLTAMOD=0
                                        ; BLTDMOD=40-36=4 come al solito

        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.w  #(64*45)+18,$58(a5)     ; BLTSIZE (via al blitter !)

        lea     2*18*45(a1),a1          ; punta al prossimo plane sorgente
                                        ; ogni plane e` largo 18 words e alto
                                        ; 45 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$475      ; color1
```

```
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 288 pixel (18 words)
; alto 45 righe e formato da 3 bitplanes

Figura:
        incbin  copmon.raw

;*****************************************************************************

        section gnippi,bss_C

BITPLANE:
                ds.b    40*256  ; 3 bitplanes
                ds.b    40*256
                ds.b    40*256

        end

;*****************************************************************************

In questo esempio facciamoun effetto "tendina", cioe' disegnamo una figura
come se fosse una tendina veneziana che viene aperta o chiusa. Fate prima
a guardarlo che a capire di cosa si tratta rileggendo la spiegazione !
Questo effetto e' ottenuto mediante una tecnica simile a quella usata
nell'esempio lezione9h3.s per disegnare un immagine una colonna per volta.
Per ottenere l'effetto si fa un AND tra la figura e un valore di maschera
che seleziona solo alcune colonne di pixel. A differenza dell'esempio
lezione9h3, non possiamo usare BLTAFWM/BLTALWM per contenere la maschera
perche' dobbiamo applicare la maschera a tutte le word della figura, non
solamente alla prima e all'ultima. Per questo facciamo un AND tra il canale
A e il canale B, teniamo disabilitato il canale B e usiamo BLTBDAT come
maschera.
La maschera viene fatta variare in modo da mostrare progressivamente tutta
l'immagine, e poi viene variata di nuovo in modo da cancellare a poco a poco
tutta l'immagine.
```

26.7 Lezione10i1

```
; Lezione10i1.s         Sine-scroller da 2 pixel
;                Tasto sinistro per uscire.

        SECTION         CiriCop,CODE

;        Include         "DaWorkBench.s"         ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include         "startup1.s"         ; Salva Copperlist Etc.
*****************************************************************************
```

Figura 26.5: Lezione 10g3

```
                ;5432109876543210
DMASET          EQU         %1000001111000000        ; copper,bitplane,blitter DMA


START:

        MOVE.L          #BITPLANE,d0         ; dove puntare
        LEA         BPLPOINTERS,A1         ; puntatori COP

        move.w          d0,6(a1)
        swap         d0
        move.w          d0,2(a1)
        swap         d0

        lea         $dff000,a5                ; CUSTOM REGISTER in a5
        MOVE.W          #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
        move.l          #COPPERLIST,$80(a5)         ; Puntiamo la nostra COP
        move.w          d0,$88(a5)                ; Facciamo partire la COP
        move.w          #0,$1fc(a5)                ; Disattiva l'AGA
        move.w          #$c00,$106(a5)                ; Disattiva l'AGA
        move.w          #$11,$10c(a5)                ; Disattiva l'AGA

        lea         testo(pc),a0                ; punta al testo dello scrolltext

mouse:
        MOVE.L          #$1ff00,d1         ; bit per la selezione tramite AND
        MOVE.L          #$10800,d2         ; linea da aspettare = $108
Waity1:
        MOVE.L          4(A5),D0         ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L          D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L          D2,D0                ; aspetta la linea $108
        BNE.S         Waity1
```

```
Waity2:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $108
        Beq.S         Waity2

        bsr.s         printchar       ; routine che stampa i nuovi chars
        bsr.s         Scorri               ; esegui la routine di scorrimento

        bsr.w         CancellaSchermo       ; pulisci lo schermo
        bsr.w         Sine               ; esegui il sine-scroll

        btst         #6,$bfe001       ; tasto sinistro del mouse premuto?
        bne.s         mouse               ; se no, torna a mouse2:
        rts

;***************************************************************************
; Questa routine stampa un carattere. Il carattere viene stampato in una
; parte di schermo invisibile.
; A0 punta al testo da stampare.
;***************************************************************************

PRINTCHAR:
        subq.w        #1,contatore        ; diminuisci il contatore di 1
        bne.s         NoPrint               ; se e‘ diverso da 0, non stampiamo,
        move.w        #16,contatore        ; altrimenti si; reinizializza il contatore

        MOVEQ         #0,D2               ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        bne.s         noreset               ; Se e‘ diverso da 0 stampalo,
        lea         testo(pc),a0        ; altrimenti ricomincia il testo daccapo
        MOVE.B        (A0)+,D2        ; Primo carattere in d2
noreset
        SUB.B         #$20,D2               ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e’ $20), in $00, quello
                                ; DELL’ASTERISCO ($21), in $01...
        ADD.L         D2,D2                ; MOLTIPLICA PER 2 IL NUMERO PRECEDENTE,
                                ; perche‘ ogni carattere e‘ largo 16 pixel
        MOVE.L        D2,A2

        ADD.L         #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        btst         #6,$02(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst         #6,$02(a5)
        bne.s         waitblit

        move.l        #$09f00000,$40(a5)        ; BLTCON0: copia da A a D
        move.l        #$ffffffff,$44(a5)        ; BLTAFWM e BLTALWM li spieghiamo dopo

        move.l        a2,$50(a5)                        ; BLTAPT: indirizzo font
        move.l        #buffer+40,$54(a5)                ; BLTDPT: indirizzo bitplane
                                                ; fisso, fuori dalla parte
                                                ; visibile dello schermo.
        move         #120-2,$64(a5)                    ; BLTAMOD: modulo font
        move         #42-2,$66(a5)                     ; BLTDMOD: modulo bit planes
        move         #(20<<6)+1,$58(a5)                ; BLTSIZE: font 16*20
NoPrint:
        rts

contatore
```

```
        dc.w        16

;***************************************************************************
; Questa routine fa scorrere il testo verso sinistra
;***************************************************************************

Scorri:

; Gli indirizzi sorgente e destinazione sono uguali.
; Shiftiamo verso sinistra, quindi usiamo il modo discendente.

        move.l      #buffer+((21*20)-1)*2,d0        ; ind. sorgente e
                                                    ; destinazione
ScorriLoop:
        btst        #6,2(a5)                ; aspetta che il blitter finisca
waitblit2:
        btst        #6,2(a5)
        bne.s        waitblit2

        move.l      #$19f00002,$40(a5)        ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l      #$ffff7fff,$44(a5)        ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $7fff = %0111111111111111
                                        ;   cancella il bit piu‘ a sinistra
; carica i puntatori

        move.l      d0,$50(a5)                      ; bltapt - sorgente
        move.l      d0,$54(a5)                      ; bltdpt - destinazione

; facciamo scorrere un immagine larga tutto lo schermo, quindi
; il modulo e‘ azzerato.

        move.l      #$00000000,$64(a5)              ; bltamod e bltdmod
        move.w      #(20*64)+21,$58(a5)              ; bltsize
                                                ; altezza 20 linee, largo 21
        rts                                        ; words (tutto lo schermo)


;***************************************************************************
; Questa routine realizza l'effetto sine-scroll. Attenzione a BLTALWM, perche'
; e' il registro dove ogni volta selezioniamo la "fettina" o "striscina"
; verticale su cui operare.
;***************************************************************************

;          ,-~~-.___.
;         / |  '     \
;        (  )         0
;         \_/-, ,----'
;            ====           //
;           /  \-'~;    /~~~(O)
;          /  __/~|   /       |
;        =(  _____| (_________|   W<

Sine:
        lea         buffer,a2               ; puntatore al buffer contenente
                                        ; lo scrolltext
        lea         bitplane,a1                ; puntatore alla destinazione

        lea         Sinustab(pc),a3                ; indirizzo tabella seno, con valori
                                        ; gia' moltiplicati * 42, per poterli
```

```
                                    ; addizionare direttamente all'address
                                    ; del bitplane.

        move.w        #$C000,d5              ; valore iniziale maschera
        moveq        #20-1,d6               ; ripeti per tutte le word dello schero
FaiUnaWord:
        moveq        #8-1,d7                      ; routine da 2 pixel. Per ogni word
                                    ; ci sono 8 "fettine" da 2 pixel
FaiUnaColonna:
        move.w        (a3)+,d0               ; legge un valore dalla tabella
        cmp.l        #EndSinustab,a3              ; se siamo alla fine della tabella
        blo.s        nostartsine               ; ricomincia da capo
        lea        sinustab(pc),a3
nostartsine:
        move.l        a1,a4                       ; copia indirizzo bitplane
        add.w        d0,a4                       ; aggiunge la coordinata Y, offset
                                    ; preso dalla sintab...

        btst        #6,2(a5)               ; aspetta che il blitter finisca
waitblit_sine:
        btst        #6,2(a5)
        bne.s        waitblit_sine

        move.w        #$ffff,$44(a5)               ; BLTAFWM
        move.w        d5,$46(a5)               ; BLTALWM - contiene la maschera che
                                    ; seleziona le "fettine" di scrolltext

        move.l        #$0bfa0000,$40(a5)        ; BLTCON0/BLTCON1 - attiva A,C,D
                                    ; D=A OR C

        move.w        #$0028,$60(a5)               ; BLTCMOD=42-2=$28
        move.l        #$00280028,$64(a5)        ; BLTAMOD=42-2=$28
                                    ; BLTDMOD=42-2=$28

        move.l        a2,$50(a5)               ; BLTAPT  (al buffer)
        move.l        a4,$48(a5)               ; BLTCPT  (allo schermo)
        move.l        a4,$54(a5)               ; BLTDPT  (allo schermo)
        move.w        #(64*20)+1,$58(a5)        ; BLTSIZE (blitta un rettangolo
                                    ; alto 20 righe e largo 1 word)

        ror.w        #2,d5                        ; spostati alla "fettina" successiva
                                    ; va a destra e dopo l'ultima "fettina"
                                    ; di una word ricomincia dalla prima
                                    ; della word seguente.
                                    ; per lo scroll da 2 pixel ogni
                                    ; "fettina" e' larga 2 pixel

        dbra        d7,FaiUnaColonna

        addq.w        #2,a2                       ; punta alla word seguente
        addq.w        #2,a1                       ; punta alla word seguente
        dbra        d6,FaiUnaWord
        rts

; Questo e' il testo. con lo 0 si termina. Il font usato ha solo i caratteri
; maiuscoli, attenzione!

testo:
        dc.b        " ECCO COME SINUSCROLLARE... IL FONT E' DI 16*20 PIXEL!..."
        dc.b        " LO SCROLL AVVIENE CON TRANQUILLITA'...",0
        even
```

```
;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
; Viene cancellata solo la parte di schermo sulla quale scorre il testo:
; dalla riga 130 alla riga 193
;***************************************************************************

CancellaSchermo:
        btst        #6,2(a5)
WBlit3:
        btst        #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s        wblit3

        move.l        #$01000000,$40(a5)        ; BLTCON0 e BLTCON1: Cancella
        move        #$0000,$66(a5)                ; BLTDMOD=0
        move.l        #bitplane+42*130,$54(a5)        ; BLTDPT
        move.w        #(64*63)+20,$58(a5)        ; BLTSIZE (via al blitter !)
                                    ; cancella dalla riga 130
                                    ; fino alla riga 193
        rts

;***************************************************************************
; Questa e' la tabella che contiene i valori delle posizioni verticali
; dello scrolltext. Le posizioni sono gia' moltiplicate per 42, quindi
; possono essere addizionate direttamente all'indirizzo del BITPLANE
;***************************************************************************

Sinustab:
        DC.W        $18C6,$191A,$1944,$1998,$19EC,$1A16,$1A6A,$1A94,$1AE8,$1B12
        DC.W        $1B3C,$1B66,$1B90,$1BBA,$1BBA,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4
        DC.W        $1BBA,$1BBA,$1B90,$1B66,$1B3C,$1B12,$1AE8,$1A94,$1A6A,$1A16
        DC.W        $19EC,$1998,$1944,$191A,$18C6,$1872,$181E,$17F4,$17A0,$174C
        DC.W        $1722,$16CE,$16A4,$1650,$1626,$15FC,$15D2,$15A8,$157E,$157E
        DC.W        $1554,$1554,$1554,$1554,$1554,$157E,$157E,$15A8,$15D2,$15FC
        DC.W        $1626,$1650,$16A4,$16CE,$1722,$174C,$17A0,$17F4,$181E,$1872
EndSinustab:

;***************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38                ; DdfStart
        dc.w        $94,$d0                ; DdfStop
        dc.w        $102,0                ; BplCon1
        dc.w        $104,0                ; BplCon2

        dc.w        $108,2                ; Il bitplane e' largo 42 bytes, ma solo 40
                                ; bytes sono visibili, quindi il modulo
                                ; vale 42-40=2
;        dc.w        $10a,2                ; Usiamo un solo bitplane, quindi BPLMOD2
                                ; non e' necessario

        dc.w        $100,$1200        ; bplcon0 - 1 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$f50        ; color1
```

```
        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;***************************************************************************

; Qui e' memorizzato il FONT di caratteri 16x20

FONT:
        incbin        "font16x20.raw"

;***************************************************************************

        SECTION        PLANEVUOTO,BSS_C

BITPLANE:
        ds.b        42*256                ; bitplane azzerato lowres

Buffer:
        ds.b        42*20                ; buffer invisibile dove viene scrollato
                                ; il testo
        end

;***************************************************************************
```

In questo esempio vediamo un sine-scroll da 2 pixel. Le routine che stampano
i caratteri e che scrollano il testo sono le stesse della lezione9n1.s,
solo che disegnano in un buffer invisibile invece che sullo schermo.
La routine "Sine" realizza l'effetto. Essa blitta tutto il contenuto del buffer
sullo schermo, a "fettine" di 2 pixel. Per questo deve eseguire blittate larghe
1 word. Ogni colonna word contiene 8 "fettine" larghe 2 pixel. Quindi la
routine ha 2 cicli annidati: quello piu' interno effettua 8 bittate per copiare
tutta la colonna di word, mentre quello piu' esterno ripete quello interno per
tutte le colonne di word dello schermo. Per selezionare le "fettine"
all'interno della word si usa una maschera contenuta in D5. Dopo ogni blittata
la maschera viene fatta ruotare in modo da selezionare ogni volta una diversa
"fettina". Poiche' con la rotazione i bit che escono da un lato del registro
rientrano dall'altro, non c'e' bisogno di settare D5 ogni volta che i bit
ad 1 escono da destra.
Ogni fettina viene copiata ad una posizione y differente, letta da una tabella
sinusiodale.
La routine di cancellazione dello schermo cancella solo la parte interessata
dal disegno. Per calcolare quale sia la parte interessata, si devono
considerare la minima e la massima coordinata Y delle "fettine", e inoltre
che le "fettine" sono alte 20 pixel. Quindi la prima riga da cancellare e'
la minima Y delle "fettine", mentre l'ultima riga e' data dalla somma della
massima Y e dell'altezza delle "fettine".

Lezione10i2

```
; Lezione10i2.s        Sine-scroller da 1 pixel
;                Tasto sinistro per uscire.

        SECTION        CiriCop,CODE

;        Include        "DaWorkBench.s"        ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include        "startup1.s"        ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
```

```
DMASET        EQU        %1000001111000000        ; copper,bitplane,blitter DMA


START:

        MOVE.L        #BITPLANE,d0        ; dove puntare
        LEA        BPLPOINTERS,A1        ; puntatori COP

        move.w        d0,6(a1)
        swap        d0
        move.w        d0,2(a1)
        swap        d0

        lea        $dff000,a5                ; CUSTOM REGISTER in a5
        MOVE.W        #DMASET,$96(a5)                ; DMACON - abilita bitplane, copper
        move.l        #COPPERLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w        d0,$88(a5)                ; Facciamo partire la COP
        move.w        #0,$1fc(a5)                ; Disattiva l'AGA
        move.w        #$c00,$106(a5)                ; Disattiva l'AGA
        move.w        #$11,$10c(a5)                ; Disattiva l'AGA

        lea        testo(pc),a0                ; punta al testo dello scrolltext

mouse:
        MOVE.L        #$1ff00,d1        ; bit per la selezione tramite AND
        MOVE.L        #$10800,d2        ; linea da aspettare = $108
Waity1:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $108
        BNE.S        Waity1
Waity2:
        MOVE.L        4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L        D1,D0                ; Seleziona solo i bit della pos. verticale
        CMPI.L        D2,D0                ; aspetta la linea $108
        Beq.S        Waity2

        bsr.s        printchar        ; routine che stampa i nuovi chars
        bsr.s        Scorri                ; esegui la routine di scorrimento

        bsr.w        CancellaSchermo        ; pulisci lo schermo
        bsr.w        Sine                ; esegui il sine-scroll

        btst        #6,$bfe001        ; tasto sinistro del mouse premuto?
        bne.s        mouse                ; se no, torna a mouse2:
        rts

;***************************************************************************
; Questa routine stampa un carattere. Il carattere viene stampato in una
; parte di schermo invisibile.
; A0 punta al testo da stampare.
;***************************************************************************

PRINTCHAR:
        subq.w        #1,contatore        ; diminuisci il contatore di 1
        bne.s        NoPrint                ; se e' diverso da 0, non stampiamo,
        move.w        #16,contatore        ; altrimenti si; reinizializza il contatore

        MOVEQ        #0,D2                ; Pulisci d2
        MOVE.B        (A0)+,D2        ; Prossimo carattere in d2
        bne.s        noreset                ; Se e' diverso da 0 stampalo,
        lea        testo(pc),a0        ; altrimenti ricomincia il testo daccapo
```

```
        MOVE.B        (A0)+,D2        ; Primo carattere in d2
noreset
        SUB.B        #$20,D2                ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                  ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                  ; DELLO SPAZIO (che e' $20), in $00, quello
                                  ; DELL'ASTERISCO ($21), in $01...
        ADD.L        D2,D2                ; MOLTIPLICA PER 2 IL NUMERO PRECEDENTE,
                                  ; perche‘ ogni carattere e‘ largo 16 pixel
        MOVE.L        D2,A2

        ADD.L        #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        btst        #6,$02(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst        #6,$02(a5)
        bne.s        waitblit

        move.l        #$09f00000,$40(a5)        ; BLTCON0: copia da A a D
        move.l        #$ffffffff,$44(a5)        ; BLTAFWM e BLTALWM li spieghiamo dopo

        move.l        a2,$50(a5)                        ; BLTAPT: indirizzo font
        move.l        #buffer+40,$54(a5)                ; BLTDPT: indirizzo bitplane
                                                ; fisso, fuori dalla parte
                                                ; visibile dello schermo.
        move        #120-2,$64(a5)                        ; BLTAMOD: modulo font
        move        #42-2,$66(a5)                        ; BLTDMOD: modulo bit planes
        move        #(20<<6)+1,$58(a5)                ; BLTSIZE: font 16*20
NoPrint:
        rts

contatore
        dc.w        16

;***************************************************************************
; Questa routine fa scorrere il testo verso sinistra
;***************************************************************************

Scorri:

; Gli indirizzi sorgente e destinazione sono uguali.
; Shiftiamo verso sinistra, quindi usiamo il modo discendente.

        move.l        #buffer+((21*20)-1)*2,d0        ; ind. sorgente e
                                                ; destinazione
ScorriLoop:
        btst        #6,2(a5)                ; aspetta che il blitter finisca
waitblit2:
        btst        #6,2(a5)
        bne.s        waitblit2

        move.l        #$19f00002,$40(a5)        ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l        #$ffff7fff,$44(a5)        ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $7fff = %0111111111111111
                                        ;   cancella il bit piu‘ a sinistra

; carica i puntatori

        move.l        d0,$50(a5)                        ; bltapt - sorgente
        move.l        d0,$54(a5)                        ; bltdpt - destinazione
```

```
; facciamo scorrere un immagine larga tutto lo schermo, quindi
; il modulo e‘ azzerato.

        move.l        #$00000000,$64(a5)               ; bltamod e bltdmod
        move.w        #(20*64)+21,$58(a5)               ; bltsize
                                                ; altezza 20 linee, largo 21
        rts                                        ; words (tutto lo schermo)


;***************************************************************************
; Questa routine realizza l'effetto sine-scroll. Attenzione a BLTALWM, perche'
; e' il registro dove ogni volta selezioniamo la "fettina" o "striscina"
; verticale su cui operare. Qua ci sono le differenze con il sine da 2 pixel!
;***************************************************************************

;          ,-~~-.___.
;         / |  '     \
;        (  )         0
;         \_/-, ,----'
;            ====           //
;           /  \-'~;    /~~~(O)
;          /  __/~|   /       |
;        =(  _____| (_________|   W<

Sine:
        lea        buffer,a2                ; puntatore al buffer contenente
                                        ; lo scrolltext
        lea        bitplane,a1                ; puntatore alla destinazione

        move.l        SinusPtr(pc),a3                ; indirizzo primo valore seno (*42)
        subq.w        #2,a3                        ; modifica primo valore
        cmp.l        #Sinustab,a3                ; se siamo all'inizio della tabella
        bhs.s        nostartptr                ; ricomincia dalla fine
        lea        EndSinustab(pc),a3
nostartptr:
        move.l        a3,SinusPtr                ; memorizza primo valore usato

        move.w        #$8000,d5                ; valore iniziale maschera
        moveq        #20-1,d6                ; ripeti per tutte le word dello schero
FaiUnaWord:
        moveq        #16-1,d7                ; routine da 1 pixel. Per ogni word
                                        ; ci sono 16 "fettine" da 1 pixel

FaiUnaColonna:
        move.w        (a3)+,d0                ; legge un valore dalla tabella
        cmp.l        #EndSinustab,a3                ; se siamo alla fine della tabella
        blo.s        nostartsine                ; ricomincia da capo
        lea        sinustab(pc),a3
nostartsine:
        move.l        a1,a4                        ; copia indirizzo bitplane
        add.w        d0,a4                        ; aggiunge la coordnata Y

        btst        #6,2(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit_sine:
        btst        #6,2(a5)
        bne.s        waitblit_sine

        move.w        #$ffff,$44(a5)                ; BLTAFWM
        move.w        d5,$46(a5)                ; BLTALWM - contiene la maschera che
                                        ; seleziona le "fettine" di scrolltext
```

```
        move.l        #$0bfa0000,$40(a5)        ; BLTCON0/BLTCON1 - attiva A,C,D
                                        ; D=A OR C

        move.w        #$0028,$60(a5)               ; BLTCMOD=42-2=$28
        move.l        #$00280028,$64(a5)        ; BLTAMOD=42-2=$28
                                        ; BLTDMOD=42-2=$28

        move.l        a2,$50(a5)                ; BLTAPT  (al buffer)
        move.l        a4,$48(a5)                ; BLTCPT  (allo schermo)
        move.l        a4,$54(a5)                ; BLTDPT  (allo schermo)
        move.w        #(64*20)+1,$58(a5)        ; BLTSIZE (blitta un rettangolo
                                        ; alto 20 righe e largo 1 word)

        ror.w        #1,d5                        ; spostati alla "fettina" successiva
                                        ; va a destra e dopo l'ultima "fettina"
                                        ; di una word ricomincia dalla prima
                                        ; della word seguente.
                                        ; per lo scroll da 1 pixel ogni
                                        ; "fettina" e' larga 1 pixel

        dbra        d7,FaiUnaColonna

        addq.w        #2,a2                        ; punta alla word seguente
        addq.w        #2,a1                        ; punta alla word seguente
        dbra        d6,FaiUnaWord
        rts

; Questo e' il testo. con lo 0 si termina. Il font usato ha solo i caratteri
; maiuscoli, attenzione!

testo:
        dc.b        " ECCO COME SINUSCROLLARE... IL FONT E' DI 16*20 PIXEL!..."
        dc.b        " LO SCROLL AVVIENE CON TRANQUILLITA'...",0
        even

;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
; Viene cancellata solo la parte di schermo sulla quale scorre il testo:
; dalla riga 130 alla riga 193
;***************************************************************************

CancellaSchermo:
        btst        #6,2(a5)
WBlit3:
        btst        #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s        wblit3

        move.l        #$01000000,$40(a5)        ; BLTCON0 e BLTCON1: Cancella
        move        #$0000,$66(a5)                ; BLTDMOD=0
        move.l        #bitplane+42*130,$54(a5)        ; BLTDPT
        move.w        #(64*63)+20,$58(a5)        ; BLTSIZE (via al blitter !)
                                        ; cancella dalla riga 130
                                        ; fino alla riga 193
        rts

;***************************************************************************

; questo puntatore contiene l'indirizzo del primo valore da leggere dalla
; tabella

SinusPtr:        dc.l        Sinustab
```

```
; Questa e' la tabella che contiene i valori delle posizioni verticali
; dello scrolltext. Le posizioni sono gia' moltiplicate per 42, quindi
; possono essere addizionate direttamente all'indirizzo del BITPLANE

Sinustab:
        DC.W        $189C,$18C6,$18F0,$191A,$1944,$196E,$1998,$19C2,$19C2,$19EC
        DC.W        $1A16,$1A40,$1A6A,$1A6A,$1A94,$1ABE,$1ABE,$1AE8,$1B12,$1B12
        DC.W        $1B3C,$1B3C,$1B66,$1B66,$1B90,$1B90,$1BBA,$1BBA,$1BBA,$1BBA
        DC.W        $1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4
        DC.W        $1BBA,$1BBA,$1BBA,$1BBA,$1B90,$1B90,$1B66,$1B66,$1B3C,$1B3C
        DC.W        $1B12,$1B12,$1AE8,$1ABE,$1ABE,$1A94,$1A6A,$1A6A,$1A40,$1A16
        DC.W        $19EC,$19C2,$19C2,$1998,$196E,$1944,$191A,$18F0,$18C6,$189C
        DC.W        $189C,$1872,$1848,$181E,$17F4,$17CA,$17A0,$1776,$1776,$174C
        DC.W        $1722,$16F8,$16CE,$16CE,$16A4,$167A,$167A,$1650,$1626,$1626
        DC.W        $15FC,$15FC,$15D2,$15D2,$15A8,$15A8,$157E,$157E,$157E,$157E
        DC.W        $1554,$1554,$1554,$1554,$1554,$1554,$1554,$1554,$1554,$1554
        DC.W        $157E,$157E,$157E,$157E,$15A8,$15A8,$15D2,$15D2,$15FC,$15FC
        DC.W        $1626,$1626,$1650,$167A,$167A,$16A4,$16CE,$16CE,$16F8,$1722
        DC.W        $174C,$1776,$1776,$17A0,$17CA,$17F4,$181E,$1848,$1872,$189C
EndSinustab:

;***************************************************************************

        SECTION        GRAPHIC,DATA_C

COPPERLIST:
        dc.w        $8E,$2c81        ; DiwStrt
        dc.w        $90,$2cc1        ; DiwStop
        dc.w        $92,$38               ; DdfStart
        dc.w        $94,$d0               ; DdfStop
        dc.w        $102,0               ; BplCon1
        dc.w        $104,0               ; BplCon2

        dc.w        $108,2               ; Il bitplane e' largo 42 bytes, ma solo 40
                                ; bytes sono visibili, quindi il modulo
                                ; vale 42-40=2
;        dc.w        $10a,2               ; Usiamo un solo bitplane, quindi BPLMOD2
                                ; non e' necessario

        dc.w        $100,$1200        ; bplcon0 - 1 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo         bitplane

        dc.w        $0180,$000        ; color0
        dc.w        $0182,$f50        ; color1

        dc.w        $FFFF,$FFFE        ; Fine della copperlist

;***************************************************************************

; Qui e' memorizzato il FONT di caratteri 16x20

FONT:
        incbin        "font16x20.raw"

;***************************************************************************

        SECTION        PLANEVUOTO,BSS_C

BITPLANE:
        ds.b        42*256               ; bitplane azzerato lowres
```

```
Buffer:
        ds.b        42*20               ; buffer invisibile dove viene scrollato
                                ; il testo
        end

;***************************************************************************

In questo esempio vediamo un sine-scroll da 1 pixel. Se guardate attentamente
i bordi delle lettere noterete come la sinusiode sia molto meno scalettata.
La routine "Sine" e' molto simile a quella di lezione10i1.s. Le differenze
sono costituite dal fatto che ora abbiamo "fettine" larghe un pixel, il che
vuol dire che in una word ci sono 16 "fettine" invece che 8, e che naturalmente
la maschera deve ruotare ogni volta di 1 pixel invece che di 2.
Notate inoltre che usiamo una tabella di seni diversa. Infatti se avessimo
usato la stessa la sinusiode sarebbe venuta piu' stretta.
Per rendere un po' piu' vario lo scroller, leggiamo i valori della tabella
ogni volta partendo da un valore diverso. Per questo usiamo un puntatore al
primo valore da leggere che viene modificato ogni volta che viene letto.
```

Figura 26.6: Lezione 10i2

26.8 Lezione10h1

```
; Lezione10h1.s Collisione tra BOB e sfondo mediante il flag ZERO del blitter.
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
```

```
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #3-1,D1         ; numero di bitplanes (qua sono 3)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0              ; + LUNGHEZZA DI UNA PLANE !!!!!
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        bsr.s   MuoviOggetto            ; sposta il bob
        bsr.w   ControllaCollisione     ; controlla e segnala eventuali
                                        ; collisioni tra bob e sfondo

        bsr.w   SalvaSfondo             ; salva lo sfondo
        bsr.w   DisegnaOggetto          ; disegna il bob

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   RipristinaSfondo        ; ripristina lo sfondo

        btst    #6,$bfe001              ; tasto sinistro del mouse premuto?
        bne.s   mouse                   ; se no, torna a mouse:

        rts


;*****************************************************************************
; Questa routine muove il bob sullo schermo.
;*****************************************************************************

MuoviOggetto
        addq.w  #1,ogg_y        ; sposta in basso il bob
        cmp.w   #256-11,ogg_y   ; e' arrivato al bordo basso ?
        bls.s   EndMuovi        ; se no fine
```

```
        clr.w   ogg_y           ; altrimenti riparti dall'alto
EndMuovi
        rts

;***************************************************************************
; Questa routine disegna il BOB alle coordinate specificate nelle variabili
; X_OGG e Y_OGG.
;***************************************************************************

;        ||||||||
;        | =  = |
;       @| O  O |@
;        |  ()  |
;        ((\__/))
;         ((()))
;          ))((
;           ()

DisegnaOggetto:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        move.w  d0,d2

        or.w    #$0FCA,d0       ; ...giusti per inserirsi nel registro BLTCON0
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     figura,a1       ; puntatore sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit2

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0 (usa A+D)
        move.w  d2,$42(a5)              ; BLTCON1 (nessun modo speciale)
        move.l  #$0022fffe,$60(a5)
        move.l  #$fffe0022,$64(a5)      ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
                                        ; BLTDMOD=40-6=34=$22 come al solito
```

```
        move.l  #Maschera,$50(a5)       ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$54(a5)              ; BLTDPT  (linee di schermo)
        move.l  a0,$48(a5)              ; BLTCPT  (linee di schermo)
        move.l  a1,$4c(a5)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     4*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni plane e' largo 2 words e alto
                                        ; 11 righe

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

;***************************************************************************
; Questa routine copia il rettangolo di sfondo che verra' sovrascritto dal
; BOB in un buffer
;***************************************************************************

SalvaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e' costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        lea     Buffer,a1       ; indirizzo destinazione
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00220000,$64(a5)      ; BLTAMOD=40-4=36=$24
                                        ; BLTDMOD=0 nel buffer
        move.l  a0,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  a1,$54(a5)              ; BLTDPT - buffer
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     40*256(a0),a0           ; punta al prossimo plane sorgente
        lea     6*11(a1),a1             ; punta al prossimo plane destinazione
                                        ; ogni blittata e' larga 3 words e alto
                                        ; 11 righe
        dbra    d7,PlaneLoop2
```

```
        rts

;***************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione.
;***************************************************************************

RipristinaSfondo:
        lea     bitplane,a0     ; destinazione in a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all’indirizzo di partenza

        move.w  ogg_x(pc),d1    ; coordinata X
        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all’indirizzo del bitplane, trovando
                                ; l’indirizzo giusto di destinazione

        lea     Buffer,a1       ; indirizzo sorgente
        moveq   #3-1,d7         ; ripeti per ogni plane
PlaneLoop3:
        btst    #6,2(a5)         ; dmaconr
WBlit4:
        btst    #6,2(a5)         ; attendi che il blitter abbia finito
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto


        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.l  #$00000022,$64(a5)      ; BLTAMOD=0 (buffer)
                                        ; BLTDMOD=40-6=34=$22
        move.l  a1,$50(a5)              ; BLTAPT (buffer)
        move.l  a0,$54(a5)              ; BLTDPT (schermo)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     40*256(a0),a0           ; punta al prossimo plane destinazione
        lea     6*11(a1),a1             ; punta al prossimo plane sorgente
                                        ; ogni blittata e‘ larga 3 words e alto
                                        ; 11 righe
        dbra    d7,PlaneLoop3
        rts

OGG_Y:          dc.w    0       ; qui viene memorizzata la Y dell’oggetto
OGG_X:          dc.w    100     ; qui viene memorizzata la X dell’oggetto


;***************************************************************************
; Questa routine controlla il verificarsi di una collisione
;***************************************************************************

ControllaCollisione
```

```
        lea     bitplane,a0     ; indirizzo bitplane a0
        move.w  ogg_y(pc),d0    ; coordinata Y
        mulu.w  #40,d0          ; calcola indirizzo: ogni riga e‘ costituita da
                                ; 40 bytes
        add.w   d0,a0           ; aggiungi all'indirizzo di partenza

        move.w  ogg_x(pc),d0    ; coordinata X
        move.w  d0,d1           ; copia
        and.w   #$000f,d0       ; si selezionano i primi 4 bit perche' vanno
                                ; inseriti nello shifter del canale A
        lsl.w   #8,d0           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d0           ; della word...
        move.w  d0,d2

        or.w    #$0AA0,d0       ; ...giusti per inserirsi nel registro BLTCON0
                                ; sono attivi solo i canali A e C (no D)
                                ; esegue un AND tra A e C

        lsr.w   #3,d1           ; (equivalente ad una divisione per 8)
                                ; arrotonda ai multipli di 8 per il puntatore
                                ; allo schermo, ovvero agli indirizzi dispari
                                ; (anche ai byte, quindi)
                                ; x es.: un 16 come coordinata diventa il
                                ; byte 2
        and.w   #$fffe,d1       ; escludo il bit 0 del
        add.w   d1,a0           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

; attende che il blitter abbia finito prima di modificare i registri
        btst    #6,2(a5)
WBlit_coll:
        btst    #6,2(a5)
        bne.s   wblit_coll

        lea     Maschera,a1     ; puntatore maschera collisione
        moveq   #3-1,d7         ; ripeti per ogni plane
CollLoop:

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word


        move.w  d0,$40(a5)              ; BLTCON0
                                        ; sono attivi solo i canali A e C
                                        ; (non D). Esegue un AND tra A e C
        move.w  #$0000,$42(a5)          ; BLTCON1 (nessun modo speciale)
        move.w  #$0022,$60(a5)          ; BLTCMOD=40-6=34=$22
        move.w  #$fffe,$64(a5)          ; BLTAMOD=$fffe=-2 torna indietro
                                        ; all'inizio della riga.
        move.l  a1,$50(a5)              ; BLTAPT  (fisso alla figura sorgente)
        move.l  a0,$48(a5)              ; BLTCPT  (posizione sullo schermo)
        move.w  #(64*11)+3,$58(a5)      ; BLTSIZE (via al blitter !)

        lea     40*256(a0),a0           ; punta al prossimo plane sullo schermo

        btst    #6,2(a5)
WBlit_coll2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit_coll2             ; prima di testare il flag ZERO

        btst    #5,2(a5)                ; testa il flag ZERO.
```

```
        beq.s   SiColl                  ; Se il flag e' diverso da zero c'e'
                                        ; stata una collisione, segnala.
                                        ; Altrimenti controlla prossimo plane

        dbra    d7,CollLoop             ; se per nessun plane si verifica la
                                        ; collisione, esci dal loop

NoColl
        move.w  #$000,$180(a5)  ; non si sono verificate collisioni:
                                ; schermo nero
        bra.s   EndColl         ; salta alla fine della routine

SiColl  move    #$F00,$180(a5)  ; e' stata rilevata una collisione:
                                ; schermo rosso

EndColl
        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; VALORE MODULO = 0
        dc.w    $10a,0          ; ENTRAMBI I MODULI ALLO STESSO VALORE.

        dc.w    $100,$3200      ; bplcon0 - 3 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000
        dc.w $e8,$0000,$ea,$0000

        dc.w    $0182,$475      ; color1
        dc.w    $0184,$fff      ; color2
        dc.w    $0186,$ccc      ; color3
        dc.w    $0188,$999      ; color4
        dc.w    $018a,$232      ; color5
        dc.w    $018c,$777      ; color6
        dc.w    $018e,$444      ; color7

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Questi sono i dati che compongono la figura del bob.
; Il bob e' in formato normale, largo 32 pixel (2 words)
; alto 11 righe e formato da 3 bitplanes

Figura: dc.l    $007fc000       ; plane 1
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
```

```
        dc.l    $0ffffe00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000

        dc.l    $00000000       ; plane 2
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $0fe07e00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000
        dc.l    $00000000

        dc.l    $007fc000       ; plane 3
        dc.l    $03803800
        dc.l    $04000400
        dc.l    $081f8200
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $10204100
        dc.l    $081f8200
        dc.l    $04000400
        dc.l    $03803800
        dc.l    $007fc000

Maschera:
        dc.l    $007fc000
        dc.l    $03fff800
        dc.l    $07fffc00
        dc.l    $0ffffe00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $1fe07f00
        dc.l    $0ffffe00
        dc.l    $07fffc00
        dc.l    $03fff800
        dc.l    $007fc000

;******************************************************************************

BITPLANE:
        incbin  "amiga.raw"             ; qua carichiamo la figura in
                                        ; formato RAWBLIT (o interleaved),
                                        ; convertita col KEFCON.

;******************************************************************************

        SECTION BUFFER,BSS_C

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 11, larghezza 3 words
; 3 bitplanes

Buffer:
        ds.w    11*3*3

        end
```

```
;*****************************************************************************

In questo esempio mostriamo come rilevare le collisioni tra il bob e lo sfondo.
Per rilevare la collisione si usa il flag Zero del blitter. La tecnica e' la
seguente: si esegue una blittata che fa un AND tra la maschera del bob e lo
sfondo, senza pero' scrivere il risultato in uscita. Se l'operazione di AND
fornisce un risultato di zero per tutti i bit della blittata, il flag Zero
assume valore 1. Il fatto che l'operazione di AND abbia fornito risultato 0,
vuol dire che in nessun caso un bit della maschera coincide con uno dello
sfondo, quindi non c'e' collisione.
Se invece almeno un bit della maschera coincide con uno dello sfondo, vuol dire
che si verifica una collisione. In questo caso, siccome facendo l'AND tra
questi 2 bit il risultato e' 1, il Flag ZERO assumera' valore 0, e da questo
fatto possiamo capire che c'e' stata una collisione.
Notate che mentre per il bob abbiamo la maschera, non e' cosi' per lo sfondo.
Questo fatto ci costringe a controllare la collisione tra la maschera del bob
e tutti i planes della figura. Disponendo della maschera dello sfondo, si
potrebbe testare la collisione mediante una sola blittata tra le maschere.
Fatelo per esercizio.
Notate anche che prima di testare il flag Zero, e' necessario attendere la
fine della blittata.
```

Figura 26.7: Lezione 10h1

Lezione10i1

```
; Lezione10i1.s Sine-scroller da 2 pixel
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP

        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     testo(pc),a0            ; punta al testo dello scrolltext

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        bsr.s   printchar       ; routine che stampa i nuovi chars
        bsr.s   Scorri          ; esegui la routine di scorrimento

        bsr.w   CancellaSchermo ; pulisci lo schermo
        bsr.w   Sine            ; esegui il sine-scroll

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse2:
        rts

;*****************************************************************************
; Questa routine stampa un carattere. Il carattere viene stampato in una
; parte di schermo invisibile.
; A0 punta al testo da stampare.
;*****************************************************************************

PRINTCHAR:
        subq.w  #1,contatore    ; diminuisci il contatore di 1
        bne.s   NoPrint         ; se e' diverso da 0, non stampiamo,
```

```
        move.w  #16,contatore   ; altrimenti si; reinizializza il contatore

        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        bne.s   noreset         ; Se e' diverso da 0 stampalo,
        lea     testo(pc),a0    ; altrimenti ricomincia il testo daccapo
        MOVE.B  (A0)+,D2        ; Primo carattere in d2
noreset
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        ADD.L   D2,D2           ; MOLTIPLICA PER 2 IL NUMERO PRECEDENTE,
                                ; perche' ogni carattere e' largo 16 pixel
        MOVE.L  D2,A2

        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst    #6,$02(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0: copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo

        move.l  a2,$50(a5)                      ; BLTAPT: indirizzo font
        move.l  #buffer+40,$54(a5)              ; BLTDPT: indirizzo bitplane
                                                ; fisso, fuori dalla parte
                                                ; visibile dello schermo.
        move    #120-2,$64(a5)                  ; BLTAMOD: modulo font
        move    #42-2,$66(a5)                   ; BLTDMOD: modulo bit planes
        move    #(20<<6)+1,$58(a5)              ; BLTSIZE: font 16*20
NoPrint:
        rts

contatore
        dc.w    16

;***************************************************************************
; Questa routine fa scorrere il testo verso sinistra
;***************************************************************************

Scorri:

; Gli indirizzi sorgente e destinazione sono uguali.
; Shiftiamo verso sinistra, quindi usiamo il modo discendente.

        move.l  #buffer+((21*20)-1)*2,d0        ; ind. sorgente e
                                                ; destinazione
ScorriLoop:
        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit2:
        btst    #6,2(a5)
        bne.s   waitblit2

        move.l  #$19f00002,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l  #$ffff7fff,$44(a5)      ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $7fff = %0111111111111111
```

```
                                        ;   cancella il bit piu‘ a sinistra
; carica i puntatori

        move.l  d0,$50(a5)                      ; bltapt - sorgente
        move.l  d0,$54(a5)                      ; bltdpt - destinazione

; facciamo scorrere un immagine larga tutto lo schermo, quindi
; il modulo e‘ azzerato.

        move.l  #$00000000,$64(a5)              ; bltamod e bltdmod
        move.w  #(20*64)+21,$58(a5)             ; bltsize
                                                ; altezza 20 linee, largo 21
        rts                                     ; words (tutto lo schermo)


;***************************************************************************
; Questa routine realizza l'effetto sine-scroll. Attenzione a BLTALWM, perche'
; e' il registro dove ogni volta selezioniamo la "fettina" o "striscina"
; verticale su cui operare.
;***************************************************************************

;         ,-~~-.___.
;        / |  '     \
;       (  )         0
;        \_/-, ,----'
;           ====           //
;          /  \-'~;    /~~~(O)
;         /  __/~|   /       |
;       =(  _____| (_________|   W<

Sine:
        lea     buffer,a2               ; puntatore al buffer contenente
                                        ; lo scrolltext
        lea     bitplane,a1             ; puntatore alla destinazione

        lea     Sinustab(pc),a3         ; indirizzo tabella seno, con valori
                                        ; gia' moltiplicati * 42, per poterli
                                        ; addizionare direttamente all'address
                                        ; del bitplane.

        move.w  #$C000,d5               ; valore iniziale maschera
        moveq   #20-1,d6                ; ripeti per tutte le word dello schero
FaiUnaWord:
        moveq   #8-1,d7                 ; routine da 2 pixel. Per ogni word
                                        ; ci sono 8 "fettine" da 2 pixel
FaiUnaColonna:
        move.w  (a3)+,d0                ; legge un valore dalla tabella
        cmp.l   #EndSinustab,a3         ; se siamo alla fine della tabella
        blo.s   nostartsine             ; ricomincia da capo
        lea     sinustab(pc),a3
nostartsine:
        move.l  a1,a4                   ; copia indirizzo bitplane
        add.w   d0,a4                   ; aggiunge la coordinata Y, offset
                                        ; preso dalla sintab...

        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit_sine:
        btst    #6,2(a5)
        bne.s   waitblit_sine

        move.w  #$ffff,$44(a5)          ; BLTAFWM
        move.w  d5,$46(a5)              ; BLTALWM - contiene la maschera che
```

```
                                    ; seleziona le "fettine" di scrolltext

        move.l  #$0bfa0000,$40(a5)      ; BLTCON0/BLTCON1 - attiva A,C,D
                                        ; D=A OR C

        move.w  #$0028,$60(a5)          ; BLTCMOD=42-2=$28
        move.l  #$00280028,$64(a5)      ; BLTAMOD=42-2=$28
                                        ; BLTDMOD=42-2=$28

        move.l  a2,$50(a5)              ; BLTAPT  (al buffer)
        move.l  a4,$48(a5)              ; BLTCPT  (allo schermo)
        move.l  a4,$54(a5)              ; BLTDPT  (allo schermo)
        move.w  #(64*20)+1,$58(a5)      ; BLTSIZE (blitta un rettangolo
                                        ; alto 20 righe e largo 1 word)

        ror.w   #2,d5                   ; spostati alla "fettina" successiva
                                        ; va a destra e dopo l'ultima "fettina"
                                        ; di una word ricomincia dalla prima
                                        ; della word seguente.
                                        ; per lo scroll da 2 pixel ogni
                                        ; "fettina" e' larga 2 pixel

        dbra    d7,FaiUnaColonna

        addq.w  #2,a2                   ; punta alla word seguente
        addq.w  #2,a1                   ; punta alla word seguente
        dbra    d6,FaiUnaWord
        rts

; Questo e' il testo. con lo 0 si termina. Il font usato ha solo i caratteri
; maiuscoli, attenzione!

testo:
        dc.b    " ECCO COME SINUSCROLLARE... IL FONT E' DI 16*20 PIXEL!..."
        dc.b    " LO SCROLL AVVIENE CON TRANQUILLITA'...",0
        even

;******************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
; Viene cancellata solo la parte di schermo sulla quale scorre il testo:
; dalla riga 130 alla riga 193
;******************************************************************************

CancellaSchermo:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move    #$0000,$66(a5)          ; BLTDMOD=0
        move.l  #bitplane+42*130,$54(a5)        ; BLTDPT
        move.w  #(64*63)+20,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; cancella dalla riga 130
                                        ; fino alla riga 193
        rts

;******************************************************************************
; Questa e' la tabella che contiene i valori delle posizioni verticali
; dello scrolltext. Le posizioni sono gia' moltiplicate per 42, quindi
; possono essere addizionate direttamente all'indirizzo del BITPLANE
;******************************************************************************
```

```
Sinustab:
        DC.W    $18C6,$191A,$1944,$1998,$19EC,$1A16,$1A6A,$1A94,$1AE8,$1B12
        DC.W    $1B3C,$1B66,$1B90,$1BBA,$1BBA,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4
        DC.W    $1BBA,$1BBA,$1B90,$1B66,$1B3C,$1B12,$1AE8,$1A94,$1A6A,$1A16
        DC.W    $19EC,$1998,$1944,$191A,$18C6,$1872,$181E,$17F4,$17A0,$174C
        DC.W    $1722,$16CE,$16A4,$1650,$1626,$15FC,$15D2,$15A8,$157E,$157E
        DC.W    $1554,$1554,$1554,$1554,$1554,$157E,$157E,$15A8,$15D2,$15FC
        DC.W    $1626,$1650,$16A4,$16CE,$1722,$174C,$17A0,$17F4,$181E,$1872
EndSinustab:

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

        dc.w    $108,2          ; Il bitplane e' largo 42 bytes, ma solo 40
                                ; bytes sono visibili, quindi il modulo
                                ; vale 42-40=2
;       dc.w    $10a,2          ; Usiamo un solo bitplane, quindi BPLMOD2
                                ; non e' necessario

        dc.w    $100,$1200      ; bplcon0 - 1 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$f50      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Qui e' memorizzato il FONT di caratteri 16x20

FONT:
        incbin  "font16x20.raw"

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    42*256          ; bitplane azzerato lowres

Buffer:
        ds.b    42*20           ; buffer invisibile dove viene scrollato
                                ; il testo
        end

;***************************************************************************
```

In questo esempio vediamo un sine-scroll da 2 pixel. Le routine che stampano i caratteri e che scrollano il testo sono le stesse della lezione9n1.s,

solo che disegnano in un buffer invisibile invece che sullo schermo.
La routine "Sine" realizza l'effetto. Essa blitta tutto il contenuto del buffer sullo schermo, a "fettine" di 2 pixel. Per questo deve eseguire blittate larghe 1 word. Ogni colonna word contiene 8 "fettine" larghe 2 pixel. Quindi la routine ha 2 cicli annidati: quello piu' interno effettua 8 bittate per copiare tutta la colonna di word, mentre quello piu' esterno ripete quello interno per tutte le colonne di word dello schermo. Per selezionare le "fettine" all'interno della word si usa una maschera contenuta in D5. Dopo ogni blittata la maschera viene fatta ruotare in modo da selezionare ogni volta una diversa "fettina". Poiche' con la rotazione i bit che escono da un lato del registro rientrano dall'altro, non c'e' bisogno di settare D5 ogni volta che i bit ad 1 escono da destra.
Ogni fettina viene copiata ad una posizione y differente, letta da una tabella sinusiodale.
La routine di cancellazione dello schermo cancella solo la parte interessata dal disegno. Per calcolare quale sia la parte interessata, si devono considerare la minima e la massima coordinata Y delle "fettine", e inoltre che le "fettine" sono alte 20 pixel. Quindi la prima riga da cancellare e' la minima Y delle "fettine", mentre l'ultima riga e' data dalla somma della massima Y e dell'altezza delle "fettine".

Lezione10i2

```
; Lezione10i2.s Sine-scroller da 1 pixel
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP

        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        lea     testo(pc),a0            ; punta al testo dello scrolltext

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
```

```
        MOVE.L  #$10800,d2      ; linea da aspettare = $108
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $108
        Beq.S   Waity2

        bsr.s   printchar       ; routine che stampa i nuovi chars
        bsr.s   Scorri          ; esegui la routine di scorrimento

        bsr.w   CancellaSchermo ; pulisci lo schermo
        bsr.w   Sine            ; esegui il sine-scroll

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse2:
        rts

;***************************************************************************
; Questa routine stampa un carattere. Il carattere viene stampato in una
; parte di schermo invisibile.
; A0 punta al testo da stampare.
;***************************************************************************

PRINTCHAR:
        subq.w  #1,contatore    ; diminuisci il contatore di 1
        bne.s   NoPrint         ; se e' diverso da 0, non stampiamo,
        move.w  #16,contatore   ; altrimenti si; reinizializza il contatore

        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        bne.s   noreset         ; Se e' diverso da 0 stampalo,
        lea     testo(pc),a0    ; altrimenti ricomincia il testo daccapo
        MOVE.B  (A0)+,D2        ; Primo carattere in d2
noreset
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        ADD.L   D2,D2           ; MOLTIPLICA PER 2 IL NUMERO PRECEDENTE,
                                ; perche' ogni carattere e' largo 16 pixel
        MOVE.L  D2,A2

        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        btst    #6,$02(a5)      ; dmaconr - aspetta che il blitter finisca
waitblit:
        btst    #6,$02(a5)
        bne.s   waitblit

        move.l  #$09f00000,$40(a5)      ; BLTCON0: copia da A a D
        move.l  #$ffffffff,$44(a5)      ; BLTAFWM e BLTALWM li spieghiamo dopo

        move.l  a2,$50(a5)                      ; BLTAPT: indirizzo font
        move.l  #buffer+40,$54(a5)              ; BLTDPT: indirizzo bitplane
                                                ; fisso, fuori dalla parte
                                                ; visibile dello schermo.
        move    #120-2,$64(a5)                  ; BLTAMOD: modulo font
```

```
        move    #42-2,$66(a5)                   ; BLTDMOD: modulo bit planes
        move    #(20<<6)+1,$58(a5)              ; BLTSIZE: font 16*20
NoPrint:
        rts

contatore
        dc.w    16

;*****************************************************************************
; Questa routine fa scorrere il testo verso sinistra
;*****************************************************************************

Scorri:

; Gli indirizzi sorgente e destinazione sono uguali.
; Shiftiamo verso sinistra, quindi usiamo il modo discendente.

        move.l  #buffer+((21*20)-1)*2,d0        ; ind. sorgente e
                                                ; destinazione
ScorriLoop:
        btst    #6,2(a5)                ; aspetta che il blitter finisca
waitblit2:
        btst    #6,2(a5)
        bne.s   waitblit2

        move.l  #$19f00002,$40(a5)      ; BLTCON0 e BLTCON1 - copia da A a D
                                        ; con shift di un pixel

        move.l  #$ffff7fff,$44(a5)      ; BLTAFWM e BLTALWM
                                        ; BLTAFWM = $ffff - passa tutto
                                        ; BLTALWM = $7fff = %0111111111111111
                                        ;   cancella il bit piu‘ a sinistra

; carica i puntatori

        move.l  d0,$50(a5)                      ; bltapt - sorgente
        move.l  d0,$54(a5)                      ; bltdpt - destinazione

; facciamo scorrere un immagine larga tutto lo schermo, quindi
; il modulo e‘ azzerato.

        move.l  #$00000000,$64(a5)              ; bltamod e bltdmod
        move.w  #(20*64)+21,$58(a5)             ; bltsize
                                                ; altezza 20 linee, largo 21
        rts                                     ; words (tutto lo schermo)


;*****************************************************************************
; Questa routine realizza l’effetto sine-scroll. Attenzione a BLTALWM, perche’
; e’ il registro dove ogni volta selezioniamo la "fettina" o "striscina"
; verticale su cui operare. Qua ci sono le differenze con il sine da 2 pixel!
;*****************************************************************************

;         ,-~~-.___.
;        / |  ’     \
;       (  )         0
;        \_/-, ,----’
;           ====           //
;          /  \-’~;    /~~~(O)
;         /  __/~|   /       |
;       =(  _____| (_________|   W<
```

```
Sine:
        lea     buffer,a2               ; puntatore al buffer contenente
                                        ; lo scrolltext
        lea     bitplane,a1             ; puntatore alla destinazione

        move.l  SinusPtr(pc),a3         ; indirizzo primo valore seno (*42)
        subq.w  #2,a3                   ; modifica primo valore
        cmp.l   #Sinustab,a3            ; se siamo all'inizio della tabella
        bhs.s   nostartptr              ; ricomincia dalla fine
        lea     EndSinustab(pc),a3
nostartptr:
        move.l  a3,SinusPtr             ; memorizza primo valore usato

        move.w  #$8000,d5               ; valore iniziale maschera
        moveq   #20-1,d6                ; ripeti per tutte le word dello schero
FaiUnaWord:
        moveq   #16-1,d7                ; routine da 1 pixel. Per ogni word
                                        ; ci sono 16 "fettine" da 1 pixel

FaiUnaColonna:
        move.w  (a3)+,d0                ; legge un valore dalla tabella
        cmp.l   #EndSinustab,a3         ; se siamo alla fine della tabella
        blo.s   nostartsine             ; ricomincia da capo
        lea     sinustab(pc),a3
nostartsine:
        move.l  a1,a4                   ; copia indirizzo bitplane
        add.w   d0,a4                   ; aggiunge la coordnata Y

        btst    #6,2(a5)        ; dmaconr - aspetta che il blitter finisca
waitblit_sine:
        btst    #6,2(a5)
        bne.s   waitblit_sine

        move.w  #$ffff,$44(a5)          ; BLTAFWM
        move.w  d5,$46(a5)              ; BLTALWM - contiene la maschera che
                                        ; seleziona le "fettine" di scrolltext

        move.l  #$0bfa0000,$40(a5)      ; BLTCON0/BLTCON1 - attiva A,C,D
                                        ; D=A OR C

        move.w  #$0028,$60(a5)          ; BLTCMOD=42-2=$28
        move.l  #$00280028,$64(a5)      ; BLTAMOD=42-2=$28
                                        ; BLTDMOD=42-2=$28

        move.l  a2,$50(a5)              ; BLTAPT  (al buffer)
        move.l  a4,$48(a5)              ; BLTCPT  (allo schermo)
        move.l  a4,$54(a5)              ; BLTDPT  (allo schermo)
        move.w  #(64*20)+1,$58(a5)      ; BLTSIZE (blitta un rettangolo
                                        ; alto 20 righe e largo 1 word)

        ror.w   #1,d5                   ; spostati alla "fettina" successiva
                                        ; va a destra e dopo l'ultima "fettina"
                                        ; di una word ricomincia dalla prima
                                        ; della word seguente.
                                        ; per lo scroll da 1 pixel ogni
                                        ; "fettina" e' larga 1 pixel

        dbra    d7,FaiUnaColonna

        addq.w  #2,a2                   ; punta alla word seguente
        addq.w  #2,a1                   ; punta alla word seguente
        dbra    d6,FaiUnaWord
```

```
        rts

; Questo e' il testo. con lo 0 si termina. Il font usato ha solo i caratteri
; maiuscoli, attenzione!

testo:
        dc.b    " ECCO COME SINUSCROLLARE... IL FONT E' DI 16*20 PIXEL!..."
        dc.b    " LO SCROLL AVVIENE CON TRANQUILLITA'...",0
        even

;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
; Viene cancellata solo la parte di schermo sulla quale scorre il testo:
; dalla riga 130 alla riga 193
;***************************************************************************

CancellaSchermo:
        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move    #$0000,$66(a5)          ; BLTDMOD=0
        move.l  #bitplane+42*130,$54(a5)        ; BLTDPT
        move.w  #(64*63)+20,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; cancella dalla riga 130
                                        ; fino alla riga 193
        rts

;***************************************************************************

; questo puntatore contiene l'indirizzo del primo valore da leggere dalla
; tabella

SinusPtr:       dc.l    Sinustab

; Questa e' la tabella che contiene i valori delle posizioni verticali
; dello scrolltext. Le posizioni sono gia' moltiplicate per 42, quindi
; possono essere addizionate direttamente all'indirizzo del BITPLANE

Sinustab:
        DC.W    $189C,$18C6,$18F0,$191A,$1944,$196E,$1998,$19C2,$19C2,$19EC
        DC.W    $1A16,$1A40,$1A6A,$1A6A,$1A94,$1ABE,$1ABE,$1AE8,$1B12,$1B12
        DC.W    $1B3C,$1B3C,$1B66,$1B66,$1B90,$1B90,$1BBA,$1BBA,$1BBA,$1BBA
        DC.W    $1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4,$1BE4
        DC.W    $1BBA,$1BBA,$1BBA,$1BBA,$1B90,$1B90,$1B66,$1B66,$1B3C,$1B3C
        DC.W    $1B12,$1B12,$1AE8,$1ABE,$1ABE,$1A94,$1A6A,$1A6A,$1A40,$1A16
        DC.W    $19EC,$19C2,$19C2,$1998,$196E,$1944,$191A,$18F0,$18C6,$189C
        DC.W    $189C,$1872,$1848,$181E,$17F4,$17CA,$17A0,$1776,$1776,$174C
        DC.W    $1722,$16F8,$16CE,$16CE,$16A4,$167A,$167A,$1650,$1626,$1626
        DC.W    $15FC,$15FC,$15D2,$15D2,$15A8,$15A8,$157E,$157E,$157E,$157E
        DC.W    $1554,$1554,$1554,$1554,$1554,$1554,$1554,$1554,$1554,$1554
        DC.W    $157E,$157E,$157E,$157E,$15A8,$15A8,$15D2,$15D2,$15FC,$15FC
        DC.W    $1626,$1626,$1650,$167A,$167A,$16A4,$16CE,$16CE,$16F8,$1722
        DC.W    $174C,$1776,$1776,$17A0,$17CA,$17F4,$181E,$1848,$1872,$189C
EndSinustab:

;***************************************************************************

        SECTION GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2

        dc.w    $108,2          ; Il bitplane e' largo 42 bytes, ma solo 40
                                ; bytes sono visibili, quindi il modulo
                                ; vale 42-40=2
;       dc.w    $10a,2          ; Usiamo un solo bitplane, quindi BPLMOD2
                                ; non e' necessario

        dc.w    $100,$1200      ; bplcon0 - 1 bitplanes lowres

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$f50      ; color1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Qui e' memorizzato il FONT di caratteri 16x20

FONT:
        incbin  "font16x20.raw"

;***************************************************************************

        SECTION PLANEVUOTO,BSS_C

BITPLANE:
        ds.b    42*256          ; bitplane azzerato lowres

Buffer:
        ds.b    42*20           ; buffer invisibile dove viene scrollato
                                ; il testo
        end

;***************************************************************************
```

In questo esempio vediamo un sine-scroll da 1 pixel. Se guardate attentamente i bordi delle lettere noterete come la sinusiode sia molto meno scalettata. La routine "Sine" e' molto simile a quella di lezione10i1.s. Le differenze sono costituite dal fatto che ora abbiamo "fettine" larghe un pixel, il che vuol dire che in una word ci sono 16 "fettine" invece che 8, e che naturalmente la maschera deve ruotare ogni volta di 1 pixel invece che di 2.
Notate inoltre che usiamo una tabella di seni diversa. Infatti se avessimo usato la stessa la sinusiode sarebbe venuta piu' stretta.
Per rendere un po' piu' vario lo scroller, leggiamo i valori della tabella ogni volta partendo da un valore diverso. Per questo usiamo un puntatore al primo valore da leggere che viene modificato ogni volta che viene letto.

26.9 Lezione10l1

```
; Lezione10l1.s Animazione ciclica con il blitter
```

```
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0
        BNE.S   Waity1

        bsr.s   Animazione      ; sposta i fotogrammi nella tabella
        move.l  Frametab(pc),a0 ; Disegna il primo frame della tabella
        bsr.s   DisegnaFrame    ; disegna il frame

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse
        rts


;*****************************************************************************
; Questa routine crea l'animazione, spostando gli indirizzi dei fotogrammi
; in maniera che ogni volta il primo della tabella vada all'ultimo posto,
; mentra gli altri scorrono tutti di un posto in direzione del primo
;*****************************************************************************

Animazione:
```

```
        addq.b  #1,ContaAnim    ; queste tre istruzioni fanno si' che il
        cmp.b   #4,ContaAnim    ; fotogramma venga cambiato una volta
        bne.s   NonCambiare     ; si e 3 no.
        clr.b   ContaAnim
        LEA     FRAMETAB(PC),a0 ; tabella dei fotogrammi
        MOVE.L  (a0),d0         ; salva il primo indirizzo in d0
        MOVE.L  4(a0),(a0)      ; sposta indietro gli altri 7 indirizzi
        MOVE.L  4*2(a0),4(a0)   ; Queste istruzioni "ruotano" gli indirizzi
        MOVE.L  4*3(a0),4*2(a0) ; della tabella.
        MOVE.L  4*4(a0),4*3(a0)
        MOVE.L  4*5(a0),4*4(a0)
        MOVE.L  4*6(a0),4*5(a0)
        MOVE.L  4*7(a0),4*6(a0)
        MOVE.L  d0,4*7(a0)      ; metti l'ex primo indirizzo all'ottavo posto

NonCambiare:
        rts

ContaAnim:
        dc.w    0

; Questa e' la tabella degli indirizzi dei fotogrammi. Gli indirizzi
; presenti nella tabella vengono "ruotati" all'interno della tabella dalla
; routine Animazione, in modo che il primo nella lista sia la prima volta il
; fotogramma1, la volta dopo il Fotogramma2, poi il 3,4,5,6,7,8 e di nuovo il
; primo, ciclicamente. In questo modo basta prendere l'indirizzo che sta
; all'inizio della tabella ogni volta dopo il "rimescolamento" per avere gli
; indirizzi dei fotogrammi in sequenza.

FRAMETAB:
        DC.L    Frame1
        DC.L    Frame2
        DC.L    Frame3
        DC.L    Frame4
        DC.L    Frame5
        DC.L    Frame6
        DC.L    Frame7
        DC.L    Frame8


;*****************************************************************************
; Questa routine copia un frame di animazione sullo schermo.
; la posizione sullo schermo e le dimensioni dei frames sono costanti
; A0 - indirizzo sorgente
;*****************************************************************************

;                  ,-~~-.___.
;                 / ()=(()   \
;                (  |         0
;                 \_,\, ,----'
;            ##XXXxxxxxxx
;                   /  ___,~;
;                  /    /~|-
;                =(   ~~  |
;          /~~~~~~~~~~~~~~~~~~~~\
;         /________________________\
;        /___________________________\
;       /_____________________________\
;          |_____________________|
;          |_____________________| W<
;          |_____________________|
;          |                     |
```

```
DisegnaFrame:

        moveq   #2-1,d7                 ; numero planes
        lea     bitplane+80*40+6,a1     ; indirizzo destinazione

DisegnaLoop:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #32,$66(a5)             ; BLTDMOD (40-8=32)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*55)+4,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 4 word
                                        ; altezza 55 linee

        lea     2*4*55(a0),a0           ; punta al prossimo plane sorgente
                                        ; ogni plane e‘ largo 4 words e alto
                                        ; 55 righe

        lea     40*256(a1),a1           ; punta al prossimo plane destinazione

        dbra    d7,DisegnaLoop

        rts

;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$2200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000

        dc.w    $180,$000       ; color0
        dc.w    $182,$00b       ; color1
        dc.w    $184,$cc0       ; color2
        dc.w    $186,$b00       ; color3

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************
; Questi sono i frames che compongono l'animazione
```

```
Frame1:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffcf,$fffff800,$7fffff87,$fffffc00,$7fffff03,$fffffc00
        dc.l    $7ffffe01,$fffffc00,$fffffc00,$fffffe00,$fffff800,$7ffffe00
        dc.l    $fffff000,$3ffffe00,$ffffff87,$fffffe00,$ffffff87,$fffffe00
        dc.l    $ffffff87,$fffffe00,$ffffff87,$fffffe00,$ffffff87,$fffffe00
        dc.l    $ffffff87,$fffffe00,$ffffff87,$fffffe00,$ffffff87,$fffffe00
        dc.l    $7fffff87,$fffffc00,$7fffff87,$fffffc00,$7fffff87,$fffffc00
        dc.l    $3fffff87,$fffff800,$3fffff87,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffcf,$fffff800
        dc.l    $3fffffb7,$fffff800,$7fffff7b,$fffffc00,$7ffffefd,$fffffc00
        dc.l    $7ffffdfe,$fffffc00,$fffffbff,$7ffffe00,$fffff7ff,$bffffe00
        dc.l    $ffffefff,$dffffe00,$ffffe078,$1ffffe00,$ffffff7b,$fffffe00
        dc.l    $ffffff7b,$fffffe00,$ffffff7b,$fffffe00,$ffffff7b,$fffffe00
        dc.l    $ffffff7b,$fffffe00,$ffffff7b,$fffffe00,$ffffff7b,$fffffe00
        dc.l    $7fffff7b,$fffffc00,$7fffff7b,$fffffc00,$7fffff7b,$fffffc00
        dc.l    $3fffff7b,$fffff800,$3fffff7b,$fffff800,$3fffff03,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000


Frame2:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $7fffff80,$3ffffc00,$fffffffc0,$3ffffe00,$fffffffe0,$3ffffe00
        dc.l    $fffffff0,$3ffffe00,$fffffffe0,$3ffffe00,$fffffffc0,$3ffffe00
        dc.l    $ffffff82,$3ffffe00,$ffffff07,$3ffffe00,$fffffe0f,$bffffe00
        dc.l    $ffffffc1f,$fffffe00,$fffff83f,$fffffe00,$fffff07f,$fffffe00
        dc.l    $7fffe0ff,$fffffc00,$7ffff1ff,$fffffc00,$7ffffbff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
```

```
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7ffffe00,$1ffffc00
        dc.l    $7ffffe7f,$dffffc00,$ffffff3f,$dffffe00,$ffffff9f,$dffffe00
        dc.l    $ffffffcf,$dffffe00,$ffffff9f,$dffffe00,$ffffff3f,$dffffe00
        dc.l    $fffffe7d,$dffffe00,$fffffcf8,$dffffe00,$fffff9f2,$5ffffe00
        dc.l    $fffff3e7,$1ffffe00,$ffffe7cf,$9ffffe00,$ffffcf9f,$fffffe00
        dc.l    $7fff9f3f,$fffffc00,$7fffce7f,$fffffc00,$7fffe4ff,$fffffc00
        dc.l    $3ffff1ff,$fffff800,$3ffffbff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

Frame3:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7ffffffd,$fffffc00
        dc.l    $7ffffffc,$fffffc00,$ffffffff,$7ffffe00,$ffffffff,$3ffffe00
        dc.l    $ffffffff,$1ffffe00,$ffff8000,$0ffffe00,$ffff8000,$07fffe00
        dc.l    $ffff8000,$07fffe00,$ffff8000,$0ffffe00,$ffffffff,$1ffffe00
        dc.l    $ffffffff,$3ffffe00,$ffffffff,$7ffffe00,$ffffffff,$fffffe00
        dc.l    $7ffffffd,$fffffc00,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7ffffff9,$fffffc00,$7ffffffa,$fffffc00
        dc.l    $7ffffffb,$7ffffc00,$ffffffff,$bffffe00,$ffffffff,$dffffe00
        dc.l    $ffff0003,$effffe00,$ffff7fff,$f7fffe00,$ffff7fff,$fbfffe00
        dc.l    $ffff7fff,$fbfffe00,$ffff7fff,$f7fffe00,$ffff0003,$effffe00
        dc.l    $ffffffff,$dffffe00,$ffffffff,$bffffe00,$ffffffff,$7ffffe00
        dc.l    $7ffffffa,$fffffc00,$7ffffff9,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
```

```
Frame4:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7ffffbff,$fffffc00,$7ffff1ff,$fffffc00
        dc.l    $7fffe0ff,$fffffc00,$fffff07f,$fffffe00,$fffff83f,$fffffe00
        dc.l    $fffffc1f,$fffffe00,$fffffe0f,$bffffe00,$ffffff07,$3ffffe00
        dc.l    $ffffff82,$3ffffe00,$fffffffc0,$3ffffe00,$fffffffe0,$3ffffe00
        dc.l    $fffffff0,$3ffffe00,$fffffffe0,$3ffffe00,$ffffffc0,$3ffffe00
        dc.l    $7fffff80,$3ffffc00,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3ffffbff,$fffff800
        dc.l    $3ffff1ff,$fffff800,$7fffe4ff,$fffffc00,$7fffce7f,$fffffc00
        dc.l    $7fff9f3f,$fffffc00,$ffffcf9f,$fffffe00,$fffffe7cf,$9fffffe00
        dc.l    $fffff3e7,$1fffffe00,$fffff9f2,$5fffffe00,$ffffffcf8,$dfffffe00
        dc.l    $ffffffe7d,$dfffffe00,$ffffffff3f,$dfffffe00,$ffffffff9f,$dfffffe00
        dc.l    $ffffffcf,$dfffffe00,$ffffffff9f,$dfffffe00,$ffffffff3f,$dfffffe00
        dc.l    $7fffffe7f,$dfffffc00,$7fffffe00,$1fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000


Frame5:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffc3,$fffff800
        dc.l    $3fffffc3,$fffff800,$7fffffc3,$fffffc00,$7fffffc3,$fffffc00
        dc.l    $7fffffc3,$fffffc00,$fffffffc3,$fffffe00,$fffffffc3,$fffffe00
        dc.l    $fffffffc3,$fffffe00,$fffffffc3,$fffffe00,$fffffffc3,$fffffe00
        dc.l    $fffffffc3,$fffffe00,$fffffffc3,$fffffe00,$fffffffc3,$fffffe00
        dc.l    $fffff800,$1ffffe00,$fffffc00,$3ffffe00,$fffffe00,$7ffffe00
        dc.l    $7fffff00,$fffffc00,$7fffff81,$fffffc00,$7fffffc3,$fffffc00
        dc.l    $3fffffe7,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
```

```
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffff81,$fffff800,$3fffffbd,$fffff800
        dc.l    $3fffffbd,$fffff800,$7fffffbd,$fffffc00,$7fffffbd,$fffffc00
        dc.l    $7fffffbd,$fffffc00,$ffffffbd,$fffffe00,$ffffffbd,$fffffe00
        dc.l    $ffffffbd,$fffffe00,$ffffffbd,$fffffe00,$ffffffbd,$fffffe00
        dc.l    $ffffffbd,$fffffe00,$ffffffbd,$fffffe00,$fffff03c,$0ffffe00
        dc.l    $fffff7ff,$effffe00,$fffffbff,$dffffe00,$fffffdff,$bffffe00
        dc.l    $7ffffeff,$7ffffc00,$7fffff7e,$fffffc00,$7fffffbd,$fffffc00
        dc.l    $3fffffdb,$fffff800,$3fffffe7,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$fffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

Frame6:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$bffffc00,$7fffffff,$1ffffc00
        dc.l    $7ffffffe,$0ffffc00,$ffffffff,$1ffffe00,$ffffffff8,$3ffffe00
        dc.l    $ffffffff0,$7ffffe00,$fffffbe0,$fffffe00,$fffff9c1,$fffffe00
        dc.l    $fffff883,$fffffe00,$fffff807,$fffffe00,$fffff80f,$fffffe00
        dc.l    $fffff81f,$fffffe00,$fffff80f,$fffffe00,$fffff807,$fffffe00
        dc.l    $7ffff803,$fffffc00,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$fffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$bffff800
        dc.l    $3fffffff,$1ffff800,$7ffffffe,$4ffffc00,$7ffffffc,$e7fffc00
        dc.l    $7ffffff9,$f3fffc00,$fffffff3,$e7fffe00,$fffff3e7,$cffffe00
        dc.l    $fffff1cf,$9ffffe00,$fffff49f,$3ffffe00,$fffff63e,$7ffffe00
        dc.l    $fffff77c,$fffffe00,$fffff7f9,$fffffe00,$fffff7f3,$fffffe00
        dc.l    $fffff7e7,$fffffe00,$fffff7f3,$fffffe00,$fffff7f9,$fffffe00
        dc.l    $7ffff7fc,$fffffc00,$7ffff000,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

Frame7:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
```

```
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $7fffff7f,$fffffc00,$fffffe7f,$fffffe00,$fffffc7f,$fffffe00
        dc.l    $fffff87f,$fffffe00,$fffff07f,$fffffe00,$ffffe000,$03fffe00
        dc.l    $ffffc000,$03fffe00,$ffffc000,$03fffe00,$ffffe000,$03fffe00
        dc.l    $fffff07f,$fffffe00,$fffff87f,$fffffe00,$fffffc7f,$fffffe00
        dc.l    $7ffffe7f,$fffffc00,$7fffff7f,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7fffff3f,$fffffc00
        dc.l    $7ffffebf,$fffffc00,$fffffdbf,$fffffe00,$fffffbbf,$fffffe00
        dc.l    $fffff7bf,$fffffe00,$ffffef80,$01fffe00,$ffffdfff,$fdfffe00
        dc.l    $ffffbfff,$fdfffe00,$ffffbfff,$fdfffe00,$ffffdfff,$fdfffe00
        dc.l    $ffffef80,$01fffe00,$fffff7bf,$fffffe00,$fffffbbf,$fffffe00
        dc.l    $7ffffdbf,$fffffc00,$7ffffebf,$fffffc00,$7fffff3f,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

Frame8:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $7ffff803,$fffffc00,$fffff807,$fffffe00,$fffff80f,$fffffe00
        dc.l    $fffff81f,$fffffe00,$fffff80f,$fffffe00,$fffff807,$fffffe00
        dc.l    $fffff883,$fffffe00,$fffff9c1,$fffffe00,$fffffbe0,$fffffe00
        dc.l    $fffffff0,$7ffffe00,$fffffff8,$3ffffe00,$fffffffc,$1ffffe00
        dc.l    $7ffffffe,$0fffcc00,$7fffffff,$1fffcc00,$7fffffff,$bfffcc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
```

```
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7ffff000,$fffffc00
        dc.l    $7ffff7fc,$fffffc00,$fffff7f9,$fffffe00,$fffff7f3,$fffffe00
        dc.l    $fffff7e7,$fffffe00,$fffff7f3,$fffffe00,$fffff7f9,$fffffe00
        dc.l    $fffff77c,$fffffe00,$fffff63e,$7ffffe00,$fffff49f,$3ffffe00
        dc.l    $fffff1cf,$9ffffe00,$fffff3e7,$cffffe00,$fffffff3,$e7fffe00
        dc.l    $7fffffc9,$f3fffc00,$7fffffc,$e7fffc00,$7fffffe,$4ffffc00
        dc.l    $3fffffff,$1ffff800,$3fffffff,$bffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

;******************************************************************************

        SECTION bitplane,BSS_C

BITPLANE:
        ds.b    40*256          ; 2 bitplanes
        ds.b    40*256

;******************************************************************************

        end
```

In questo esempio mostriamo un animazione realizzata con il blitter.
Abbiamo 8 fotogrammi che si ripetono ciclicamente. Per realizzare l'animazione
e' sufficente disegnare con il blitter i vari fotogrammi. In questo esempio
i vari fotogrammi hanno tutti le stesse dimensioni e vengono disegnati nella
stessa posizione sullo schermo, cosa che facilita la realizzazione della
routine di disegno. Per scegliere quale sia ogni volta il fotogramma da
disegnare si impiega una routine molto simile a quella usata nell'esempio
lezione7z.s per gli sprite animati. Abbiamo una tabella con gli indirizzi
dei vari fotogrammi. Gli indirizzi vengono di volta in volta ruotati nella
tabella in modo che al primo posto venga sempre a trovarsi il fotogramma
da disegnare. Inoltre c'e' un contatore (ContaAnim) che ci permette di
disegnare il nuovo frame non tutte le volte che la routine viene eseguita ma
solo alcune. Questo contatore viene incrementato ogni volta e solo quando
raggiunge un particolare valore, gli indirizzi dei frame vengono ruotati.
Modificando questo valore, si puo' controllare la velocita' dell'animazione.

Lezione10l2

```
; Lezione10l2.s Animazione "avanti/indietro" con il blitter
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

******************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
******************************************************************************

                ;5432109876543210
```

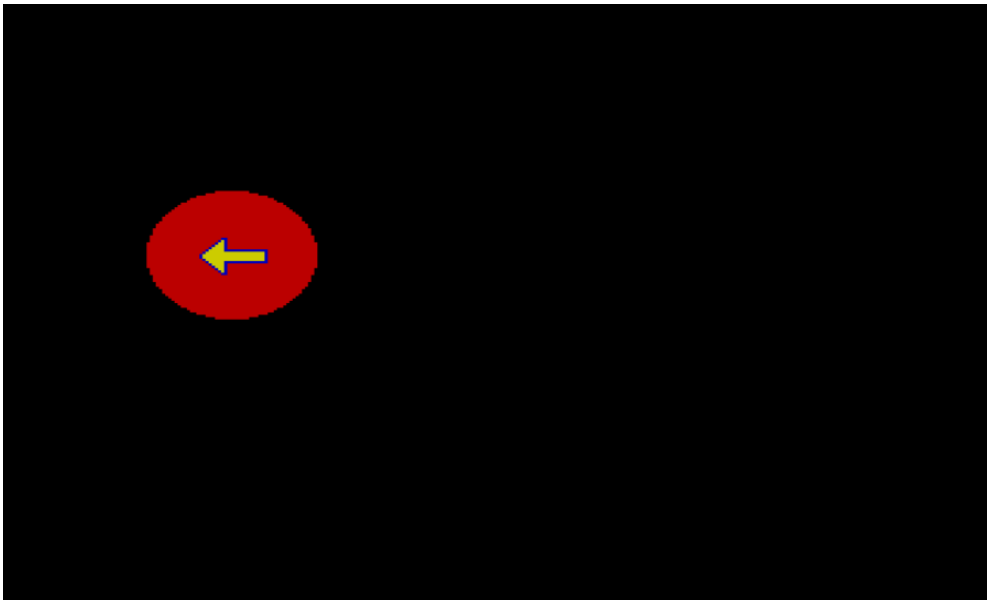


Figura 26.8: Lezione 10l1

```
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0
```

```
        BNE.S   Waity1

        bsr.s   Animazione      ; sposta i fotogrammi nella tabella
        move.l  Frametab(pc),a0 ; Disegna il primo frame della tabella
        bsr.w   DisegnaFrame    ; disegna il frame

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse
        rts


;*****************************************************************************
; Questa routine crea l'animazione, spostando gli indirizzi dei fotogrammi.
; Gli indirizzi scorrono in avanti o all'indietro a seconda della direzione
; dell'animazione.
;*****************************************************************************

Animazione:
        addq.b  #1,ContaAnim    ; queste tre istruzioni fanno si' che il
        cmp.b   #10,ContaAnim   ; fotogramma venga cambiato una volta
        bne.w   NonCambiare     ; si e 9 no.
        clr.b   ContaAnim

        tst.b   Direzione       ; controlla flag direzione
        beq.s   Avanti          ; se flag=0 va in avanti

        LEA     FRAMETAB(PC),a0 ; tabella dei fotogrammi
        MOVE.L  4*4(a0),d0      ; salva l'ultimo indirizzo in d0

        MOVE.L  4*3(a0),4*4(a0) ; sposta avanti gli altri indirizzi
        MOVE.L  4*2(a0),4*3(a0) ; Queste istruzioni "ruotano" gli indirizzi
        MOVE.L  4(a0),4*2(a0)   ; della tabella.
        MOVE.L  (a0),4(a0)
        MOVE.L  d0,(a0)         ; metti l'ex ultimo indirizzo al primo posto

        CMP.L   #FRAME1,(a0)    ; abbiamo il primo frame in testa ?
        BNE.S   AncoraIndietro  ; no, continua ad andare indietro
        CLR.B   Direzione       ; si, devi cambiare direzione
AncoraIndietro:
        BRA.S   NonCambiare

Avanti: LEA     FRAMETAB(PC),a0 ; tabella dei fotogrammi
        MOVE.L  (a0),d0         ; salva il primo indirizzo in d0

        MOVE.L  4(a0),(a0)      ; sposta indietro gli altri indirizzi
        MOVE.L  4*2(a0),4(a0)   ; Queste istruzioni "ruotano" gli indirizzi
        MOVE.L  4*3(a0),4*2(a0) ; della tabella.
        MOVE.L  4*4(a0),4*3(a0)
        MOVE.L  d0,4*4(a0)      ; metti l'ex primo indirizzo all'ottavo posto

        CMP.L   #FRAME5,(A0)    ; abbiamo il primo frame in testa ?
        BNE.S   AncoraAvanti    ; no, continua ad andare indietro
        MOVE.B  #-1,Direzione   ; si, devi cambiare direzione
AncoraAvanti:

NonCambiare:
        rts

; flag che indica la direzione dell'animazione
Direzione:
        dc.b    0
```

```
ContaAnim:
        dc.b    0

; Questa e‘ la tabella degli indirizzi dei fotogrammi. Gli indirizzi
; presenti nella tabella vengono "ruotati" all’interno della tabella dalla
; routine Animazione, in modo che il primo nella lista sia la prima volta il
; fotogramma1, la volta dopo il Fotogramma2, poi il 3,4,5 e di nuovo il
; primo, ciclicamente. In questo modo basta prendere l’indirizzo che sta
; all’inizio della tabella ogni volta dopo il "rimescolamento" per avere gli
; indirizzi dei fotogrammi in sequenza.

FRAMETAB:
        DC.L    Frame1
        DC.L    Frame2
        DC.L    Frame3
        DC.L    Frame4
        DC.L    Frame5


;***************************************************************************
; Questa routine copia un frame di animazione sullo schermo.
; la posizione sullo schermo e le dimensioni dei frames sono costanti
; A0 - indirizzo sorgente
;***************************************************************************

;                  ,-~~-.___.
;                 / ()=(()   \
;                (  |         0
;                 \_,\, ,----’
;             ##XXXxxxxxxx
;                    /  ---’~;
;                   /    /~|-
;                 =(   ~~  |
;          /~~~~~~~~~~~~~~~~~~~~\
;         /________________________\
;        /__________________________\
;       /____________________________\
;          |______________________|
;          |______________________| W<
;          |______________________|
;          |                      |

DisegnaFrame:
        moveq   #2-1,d7                 ; numero planes
        lea     bitplane+80*40+6,a1     ; indirizzo destinazione

DisegnaLoop:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.l  #$ffffffff,$44(a5)      ; maschere
        move.l  #$09f00000,$40(a5)      ; BLTCON0  e BLTCON1 (usa A+D)
                                        ; copia normale
        move.w  #0,$64(a5)              ; BLTAMOD (=0)
        move.w  #32,$66(a5)             ; BLTDMOD (40-8=32)
        move.l  a0,$50(a5)              ; BLTAPT  puntatore sorgente
        move.l  a1,$54(a5)              ; BLTDPT  puntatore destinazione
        move.w  #(64*55)+4,$58(a5)      ; BLTSIZE (via al blitter !)
                                        ; larghezza 4 word
                                        ; altezza 55 linee
```

```
        lea     2*4*55(a0),a0           ; punta al prossimo plane sorgente
                                        ; ogni plane e' largo 4 words e alto
                                        ; 55 righe

        lea     40*256(a1),a1           ; punta al prossimo plane destinazione

        dbra    d7,DisegnaLoop

        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$2200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000

        dc.w    $180,$000       ; color0
        dc.w    $182,$00b       ; color1
        dc.w    $184,$cc0       ; color2
        dc.w    $186,$b00       ; color3

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************
; Questi sono i frames che compongono l'animazione

Frame1:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3ffffffc,$fffff800,$7fffff87,$fffffc00,$7fffff03,$fffffc00
        dc.l    $7ffffe01,$fffffc00,$fffffc00,$fffffe00,$fffff800,$7ffffe00
        dc.l    $fffff000,$3ffffe00,$ffffff87,$fffffe00,$ffffff87,$fffffe00
        dc.l    $ffffff87,$fffffe00,$ffffff87,$fffffe00,$ffffff87,$fffffe00
        dc.l    $ffffff87,$fffffe00,$ffffff87,$fffffe00,$ffffff87,$fffffe00
        dc.l    $7fffff87,$fffffc00,$7fffff87,$fffffc00,$7fffff87,$fffffc00
        dc.l    $3fffff87,$fffff800,$3fffff87,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
```

```
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffcf,$fffff800
        dc.l    $3fffffb7,$fffff800,$7fffff7b,$fffffc00,$7ffffefd,$fffffc00
        dc.l    $7ffffdfe,$fffffc00,$fffffbff,$7ffffe00,$fffff7ff,$bffffe00
        dc.l    $ffffefff,$dffffe00,$fffe078,$1ffffe00,$ffffff7b,$ffffffe00
        dc.l    $ffffff7b,$ffffffe00,$ffffff7b,$ffffffe00,$ffffff7b,$ffffffe00
        dc.l    $ffffff7b,$ffffffe00,$ffffff7b,$ffffffe00,$ffffff7b,$ffffffe00
        dc.l    $7fffff7b,$fffffc00,$7fffff7b,$fffffc00,$7fffff7b,$fffffc00
        dc.l    $3fffff7b,$fffff800,$3fffff7b,$fffff800,$3fffff03,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

Frame2:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $7fffff80,$3ffffc00,$fffffffc0,$3ffffe00,$fffffffe0,$3ffffe00
        dc.l    $ffffffff0,$3ffffe00,$fffffffe0,$3ffffe00,$fffffffc0,$3ffffe00
        dc.l    $ffffff82,$3ffffe00,$ffffff07,$3ffffe00,$fffffe0f,$bffffe00
        dc.l    $fffffc1f,$fffffe00,$fffff83f,$fffffe00,$fffff07f,$fffffe00
        dc.l    $7fffe0ff,$fffffc00,$7ffff1ff,$fffffc00,$7ffffbff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7ffffe00,$1ffffc00
        dc.l    $7ffffe7f,$dffffc00,$ffffff3f,$dffffe00,$ffffff9f,$dffffe00
        dc.l    $ffffffcf,$dffffe00,$ffffff9f,$dffffe00,$ffffff3f,$dffffe00
        dc.l    $fffffe7d,$dffffe00,$fffffcf8,$dffffe00,$fffff9f2,$5ffffe00
        dc.l    $fffff3e7,$1ffffe00,$ffffe7cf,$9ffffe00,$ffffcf9f,$ffffffe00
        dc.l    $7fff9f3f,$fffffc00,$7fffce7f,$fffffc00,$7fffe4ff,$fffffc00
        dc.l    $3ffff1ff,$fffff800,$3ffffbff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

Frame3:
```

```
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7ffffffd,$fffffc00
        dc.l    $7ffffffc,$fffffc00,$ffffffff,$7ffffe00,$ffffffff,$3ffffe00
        dc.l    $ffffffff,$1ffffe00,$ffff8000,$0ffffe00,$ffff8000,$07fffe00
        dc.l    $ffff8000,$07fffe00,$ffff8000,$0ffffe00,$ffffffff,$1ffffe00
        dc.l    $ffffffff,$3ffffe00,$ffffffff,$7ffffe00,$ffffffff,$fffffe00
        dc.l    $7ffffffd,$fffffc00,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7fffffff,$fffffc00,$7ffffffa,$fffffc00
        dc.l    $7ffffffb,$7ffffc00,$ffffffff,$bfffffe00,$ffffffff,$dfffffe00
        dc.l    $ffff0003,$effffe00,$ffff7fff,$f7fffe00,$ffff7fff,$fbfffe00
        dc.l    $ffff7fff,$fbfffe00,$ffff7fff,$f7fffe00,$ffff0003,$effffe00
        dc.l    $ffffffff,$dfffffe00,$ffffffff,$bfffffe00,$ffffffff,$7ffffe00
        dc.l    $7ffffffa,$fffffc00,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000


Frame4:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $3fffffff,$fffff800,$7ffffbff,$fffffc00,$7ffff1ff,$fffffc00
        dc.l    $7fffe0ff,$fffffc00,$fffff07f,$fffffe00,$fffff83f,$fffffe00
        dc.l    $fffffc1f,$fffffe00,$fffffe0f,$bffffe00,$ffffff07,$3ffffe00
        dc.l    $ffffff82,$3ffffe00,$ffffffc0,$3ffffe00,$ffffffe0,$3ffffe00
        dc.l    $fffffff0,$3ffffe00,$ffffffe0,$3ffffe00,$ffffffc0,$3ffffe00
        dc.l    $7fffff80,$3ffffc00,$7fffffff,$fffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
```

```
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3ffffbff,$fffff800
        dc.l    $3ffff1ff,$fffff800,$7fffe4ff,$fffffc00,$7fffce7f,$fffffc00
        dc.l    $7fff9f3f,$fffffc00,$ffffcf9f,$fffffe00,$ffffe7cf,$9ffffe00
        dc.l    $fffff3e7,$1ffffe00,$fffff9f2,$5ffffe00,$fffffcf8,$dffffe00
        dc.l    $fffffe7d,$dffffe00,$ffffff3f,$dffffe00,$ffffff9f,$dffffe00
        dc.l    $ffffffcf,$dffffe00,$ffffff9f,$dffffe00,$ffffff3f,$dffffe00
        dc.l    $7ffffe7f,$dfffffc00,$7ffffe00,$1ffffc00,$7fffffff,$fffffc00
        dc.l    $3fffffff,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000


Frame5:
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffffff,$fffff800,$3fffffc3,$fffff800
        dc.l    $3fffffc3,$fffff800,$7fffffc3,$fffffc00,$7fffffc3,$fffffc00
        dc.l    $7fffffc3,$fffffc00,$ffffffc3,$fffffe00,$ffffffc3,$fffffe00
        dc.l    $ffffffc3,$fffffe00,$ffffffc3,$fffffe00,$ffffffc3,$fffffe00
        dc.l    $ffffffc3,$fffffe00,$ffffffc3,$fffffe00,$ffffffc3,$fffffe00
        dc.l    $fffff800,$1ffffe00,$fffffc00,$3ffffe00,$fffffe00,$7ffffe00
        dc.l    $7fffff00,$fffffc00,$7fffff81,$fffffc00,$7fffffc3,$fffffc00
        dc.l    $3fffffe7,$fffff800,$3fffffff,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000
        dc.l    $000003ff,$80000000,$00001fff,$f0000000,$0000ffff,$fe000000
        dc.l    $0003ffff,$ff800000,$0007ffff,$ffc00000,$001fffff,$fff00000
        dc.l    $003fffff,$fff80000,$007fffff,$fffc0000,$00ffffff,$fffe0000
        dc.l    $01ffffff,$ffff0000,$03ffffff,$ffff8000,$07ffffff,$ffffc000
        dc.l    $07ffffff,$ffffc000,$0fffffff,$ffffe000,$1fffffff,$fffff000
        dc.l    $1fffffff,$fffff000,$3fffff81,$fffff800,$3fffffbd,$fffff800
        dc.l    $3fffffbd,$fffff800,$7fffffbd,$fffffc00,$7fffffbd,$fffffc00
        dc.l    $7fffffbd,$fffffc00,$ffffffbd,$fffffe00,$ffffffbd,$fffffe00
        dc.l    $ffffffbd,$fffffe00,$ffffffbd,$fffffe00,$ffffffbd,$fffffe00
        dc.l    $ffffffbd,$fffffe00,$ffffffbd,$fffffe00,$fffff03c,$0ffffe00
        dc.l    $fffff7ff,$effffe00,$fffffbff,$dffffe00,$fffffdff,$bffffe00
        dc.l    $7ffffeff,$7fffec00,$7fffff7e,$fffffc00,$7fffffbd,$fffffc00
        dc.l    $3fffffdb,$fffff800,$3fffffe7,$fffff800,$3fffffff,$fffff800
        dc.l    $1fffffff,$fffff000,$1fffffff,$fffff000,$0fffffff,$ffffe000
        dc.l    $07ffffff,$ffffc000,$07ffffff,$ffffc000,$03ffffff,$ffff8000
        dc.l    $01ffffff,$ffff0000,$00ffffff,$fffe0000,$007fffff,$fffc0000
        dc.l    $003fffff,$fff80000,$001fffff,$fff00000,$0007ffff,$ffc00000
        dc.l    $0003ffff,$ff800000,$0000ffff,$fe000000,$00001fff,$f0000000
        dc.l    $000003ff,$80000000

;******************************************************************************

        SECTION bitplane,BSS_C
```

```
BITPLANE:
        ds.b    40*256          ; 2 bitplanes
        ds.b    40*256

;*****************************************************************************

        end
```

In questo esempio mostriamo un animazione del tipo "avanti/indierto" realizzata
con il blitter. L'animazione e' costituita da 5 fotogrammi. Questi fotogrammi
vanno mostrati in un primo momento dal primo fino all'ultimo e subito dopo
in ordine inverso fino a tornare al primo.
L'ordine dunque e' il seguente: 1-2-3-4-5-4-3-2-1-2-3- ecc.
Per realizzare quest'ordine abbiamo una routine di animazione che in base
allo stato di un flag fa scorrere gli indirizzi nella tabella in avanti o
all'indietro. Quando si il primo o l'ultimo fotogramma raggiungono il primo
posto della tabella lo stato del flag viene invertito, determinando
l'inversione della direzione di animazione.
La routine di disegno e' identica a quella di lezione10l1.s

26.10 Lezione10m1

```
; Lezione10m1.s Routine Universale Bob
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*256,d0      ; + lunghezza bitplane (qua e' alto 256 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA
```

```
mouse:

; parametri per routine SalvaSfondo

        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  #32,d2                  ; dimensione X
        move.w  #30,d3                  ; dimensione Y
        bsr.w   SalvaSfondo             ; salva lo sfondo

; parametri per routine UniBob

        move.l  Frametab(pc),a0         ; mette il puntatore al fotogramma
                                        ; da disegnare in A0
        lea     2*4*30(a0),a1           ; puntatore alla maschera in A1
        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  #32,d2                  ; dimensione X
        move.w  #30,d3                  ; dimensione Y
        bsr.w   UniBob                  ; disegna il bob con la routine
                                        ; universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0
        BNE.S   Waity1

; parametri per routine RipristinaSfondo

        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  #32,d2                  ; dimensione X
        move.w  #30,d3                  ; dimensione Y
        bsr.w   RipristinaSfondo        ; ripristina lo sfondo

        bsr.s   MuoviOggetto    ; sposta l'oggetto sullo schermo
        bsr.s   Animazione      ; sposta i fotogrammi nella tabella

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse
        rts


;*****************************************************************************
; Questa routine muove il bob sullo schermo.
;*****************************************************************************

MuoviOggetto:
        addq.w  #1,ogg_x        ; sposta in basso il bob
        cmp.w   #320-32,ogg_x   ; e' arrivato al bordo basso ?
        bls.s   EndMuovi        ; se no fine
        clr.w   ogg_x           ; altrimenti riparti dall'alto
EndMuovi
        rts

;*****************************************************************************
; Questa routine crea l'animazione, spostando gli indirizzi dei fotogrammi
; in maniera che ogni volta il primo della tabella vada all'ultimo posto,
; mentra gli altri scorrono tutti di un posto in direzione del primo
```

```
;******************************************************************************

Animazione:
        addq.b  #1,ContaAnim    ; queste tre istruzioni fanno si' che il
        cmp.b   #4,ContaAnim    ; fotogramma venga cambiato una volta
        bne.s   NonCambiare     ; si e 3 no.
        clr.b   ContaAnim
        LEA     FRAMETAB(PC),a0 ; tabella dei fotogrammi
        MOVE.L  (a0),d0         ; salva il primo indirizzo in d0
        MOVE.L  4(a0),(a0)      ; sposta indietro gli altri indirizzi
        MOVE.L  4*2(a0),4(a0)   ; Queste istruzioni "ruotano" gli indirizzi
        MOVE.L  4*3(a0),4*2(a0) ; della tabella.
        MOVE.L  d0,4*3(a0)      ; metti l'ex primo indirizzo all'ottavo posto

NonCambiare:
        rts

ContaAnim:
        dc.w    0

; Questa e' la tabella degli indirizzi dei fotogrammi. Gli indirizzi
; presenti nella tabella vengono "ruotati" all'interno della tabella dalla
; routine Animazione, in modo che il primo nella lista sia la prima volta il
; fotogramma1, la volta dopo il Fotogramma2, poi il 3, il 4 e di nuovo il
; primo, ciclicamente. In questo modo basta prendere l'indirizzo che sta
; all'inizio della tabella ogni volta dopo il "rimescolamento" per avere gli
; indirizzi dei fotogrammi in sequenza.

FRAMETAB:
        DC.L    Frame1
        DC.L    Frame2
        DC.L    Frame3
        DC.L    Frame4

; Variabili posizione BOB

OGG_Y:          dc.w    100     ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    50      ; qui viene memorizzata la X dell'oggetto

;******************************************************************************
; Questa e' la routine universale per disegnare bob di forma e dimensioni
; arbitrarie. Tutti i parametri sono passati tramite registri.
; La routine funziona su schermo normale
;
; A0 - indirizzo figura bob
; A1 - indirizzo maschera bob
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;******************************************************************************

;              ___  Oo          .:/
;             (___)o_o        ,,///;,   ,;/
;        //====--//(_)       o::::::::;;///
;                \\ ^       >:::::::::;;\\\
;                             ''\\\\\'" ';\

UniBob:

; calcolo indirizzo di partenza del blitter
```

```
        lea     bitplane,a2     ; indirizzo bitplane
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a2           ; aggiungi ad indirizzo
        move.w  d0,d6           ; copia la X
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a2           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        and.w   #$000f,d6       ; si selezionano i primi 4 bit della X perche'
                                ; vanno inseriti nello shifter dei canali A,B
        lsl.w   #8,d6           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d6           ; della word. Questo e' il valore di BLTCON1

        move.w  d6,d5           ; copia per calcolare il valore di BLTCON0
        or.w    #$0FCA,d5       ; valori da mettere in BLTCON0

; calcola offset tra i planes della figura
        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        move.w  d2,d0           ; copia larghezza divisa per 8
        mulu    d3,d2           ; moltiplica per l'altezza

; calcolo modulo blitter

        addq.w  #2,d0           ; la blittata e' una word piu' larga
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d0,d4           ; modulo=larg. schermo-larg. rettangolo

; calcolo dimensione blittata

        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d0           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d0,d3           ; metti insieme le dimensioni

; inizializza i registri che restano costanti
        btst    #6,2(a5)
WBlit_u1:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit_u1

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word

        move.w  d6,$42(a5)              ; BLTCON1 - valore di shift
                                        ; nessun modo speciale

        move.w  d5,$40(a5)              ; BLTCON0 - valore di shift
                                        ; cookie-cut

        move.l  #$fffefffe,$62(a5)      ; BLTBMOD e BLTAMOD=$fffe=-2 torna
                                        ; indietro all'inizio della riga.

        move.w  d4,$60(a5)              ; BLTCMOD valore calcolato
        move.w  d4,$66(a5)              ; BLTDMOD valore calcolato

        moveq   #2-1,d7                 ; ripeti per ogni plane
PlaneLoop:
        btst    #6,2(a5)
WBlit_u2:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
```

```
        bne.s   wblit_u2


        move.l  a1,$50(a5)              ; BLTAPT  (maschera)
        move.l  a2,$54(a5)              ; BLTDPT  (linee di schermo)
        move.l  a2,$48(a5)              ; BLTCPT  (linee di schermo)
        move.l  a0,$4c(a5)              ; BLTBPT  (figura bob)
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        add.l   d2,a0                   ; punta al prossimo plane sorgente

        lea     40*256(a2),a2           ; punta al prossimo plane destinazione
        dbra    d7,PlaneLoop

        rts

;***************************************************************************
; Questa routine copia il rettangolo di sfondo che verra' sovrascritto dal
; BOB in un buffer. La routine gestisce un bob di dimensioni arbitrarie.
; Se usate questa routine per bob di dimensioni diverse, fate attenzione
; che il buffer possa contenere il bob di dimensione massima!
; La posizione e la dimensione del rettangolo sono dei parametri
;
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;***************************************************************************

SalvaSfondo:
; calcolo indirizzo di partenza del blitter

        lea     bitplane,a1     ; indirizzo bitplane
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a1           ; aggiungi ad indirizzo
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

; calcola offset tra i planes della figura
        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        addq.w  #2,d2           ; la blittata e' larga 1 word in piu'
        move.w  d2,d0           ; copia larghezza divisa per 8
        mulu    d3,d0           ; moltiplica per l'altezza

; calcolo modulo blitter
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d2,d4           ; modulo=larg. schermo-larg. rettangolo

; calcolo dimensione blittata
        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d2,d3           ; metti insieme le dimensioni

        lea     Buffer,a2       ; indirizzo destinazione
        moveq   #2-1,d7         ; ripeti per ogni plane
PlaneLoop2:
        btst    #6,2(a5) ; dmaconr
WBlit3:
```

```
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.w  d4,$64(a5)              ; BLTAMOD valore calcolato
        move.w  #$0000,$66(a5)          ; BLTDMOD=0 nel buffer
        move.l  a1,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  a2,$54(a5)              ; BLTDPT - buffer
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        lea     40*256(a1),a1           ; punta al prossimo plane sorgente
        add.l   d0,a2                   ; punta al prossimo plane destinazione

        dbra    d7,PlaneLoop2

        rts

;*****************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione. La routine gestisce un bob di
; dimensioni arbitrarie.
; Se usate questa routine per bob di dimensioni diverse, fate attenzione
; che il buffer possa contenere il bob di dimensione massima!
; La posizione e la dimensione del rettangolo sono dei parametri
;
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;*****************************************************************************

RipristinaSfondo:
; calcolo indirizzo di partenza del blitter

        lea     bitplane,a1     ; indirizzo bitplane
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a1           ; aggiungi ad indirizzo
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

; calcola offset tra i planes della figura
        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        addq.w  #2,d2           ; la blittata e' larga 1 word in piu'
        move.w  d2,d0           ; copia larghezza divisa per 8
        mulu    d3,d0           ; moltiplica per l'altezza

; calcolo modulo blitter
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d2,d4           ; modulo=larg. schermo-larg. rettangolo

; calcolo dimensione blittata
        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d2,d3           ; metti insieme le dimensioni
```

```
        lea     Buffer,a2       ; indirizzo destinazione
        moveq   #2-1,d7         ; ripeti per ogni plane
PlaneLoop3:
        btst    #6,2(a5) ; dmaconr
WBlit4:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.w  d4,$66(a5)              ; BLTDMOD valore calcolato
        move.w  #$0000,$64(a5)          ; BLTAMOD=0 nel buffer
        move.l  a2,$50(a5)              ; BLTAPT - buffer
        move.l  a1,$54(a5)              ; BLTDPT - schermo
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        lea     40*256(a1),a1           ; punta al prossimo plane destinazione
        add.l   d0,a2                   ; punta al prossimo plane sorgente

        dbra    d7,PlaneLoop3

        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$2200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000

        dc.w    $180,$000       ; color0
        dc.w    $182,$00b       ; color1
        dc.w    $184,$cc0       ; color2
        dc.w    $186,$b00       ; color3

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************
; Questi sono i frames che compongono l'animazione

Frame1:
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00000000
        dc.l    $00000000,$00000000,$00000000,$00000000,$03ffff80,$03ffff80
        dc.l    $03ffff80,$03ffff80,$03ffff80,$03ffff80,$03ffff80,$03ffff80
        dc.l    $03ffff80,$03ffff80,$00000000,$00000000,$00000000,$00000000
```

```
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00000000
        dc.l    $00010000,$00038000,$0007c000,$000fe000,$001ff000,$003ff800
        dc.l    $007ffc00,$00fffe00,$01ffff00,$03ffff80,$03ffff80,$03ffff80
        dc.l    $03ffff80,$03ffff80,$03ffff80,$03ffff80,$03ffff80,$03ffff80
        dc.l    $03ffff80,$03ffff80,$03ffff80,$01ffff00,$00fffe00,$007ffc00
        dc.l    $003ff800,$001ff000,$000fe000,$0007c000,$00038000,$00010000
; maschera
        dc.l    $00010000,$00038000,$0007c000,$000fe000,$001ff000,$003ff800
        dc.l    $007ffc00,$00fffe00,$01ffff00,$03ffff80,$03ffff80,$03ffff80
        dc.l    $03ffff80,$03ffff80,$03ffff80,$03ffff80,$03ffff80,$03ffff80
        dc.l    $03ffff80,$03ffff80,$03ffff80,$01ffff00,$00fffe00,$007ffc00
        dc.l    $003ff800,$001ff000,$000fe000,$0007c000,$00038000,$00010000


Frame2:
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00300000
        dc.l    $00780000,$00fc0000,$01fe0000,$03ff0000,$07ff8000,$0fffc000
        dc.l    $07ffe000,$03fff000,$01fff800,$00fffc00,$007ffe00,$003fff00
        dc.l    $001fff80,$000fff00,$0007fe00,$0003fc00,$0001f800,$0000f000
        dc.l    $00006000,$00000000,$00000000,$00000000,$00000000,$00000000
        dc.l    $00000000,$00000000,$00000000,$00000000,$001fffc0,$003fffc0
        dc.l    $007fffc0,$00ffffc0,$01ffffc0,$03ffffc0,$07ffffc0,$0fffffc0
        dc.l    $0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0
        dc.l    $0fffff80,$0fffff00,$0ffffe00,$0ffffc00,$0ffff800,$0ffff000
        dc.l    $0fffe000,$00000000,$00000000,$00000000,$00000000,$00000000

        dc.l    $00000000,$00000000,$00000000,$00000000,$001fffc0,$003fffc0
        dc.l    $007fffc0,$00ffffc0,$01ffffc0,$03ffffc0,$07ffffc0,$0fffffc0
        dc.l    $0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0
        dc.l    $0fffff80,$0fffff00,$0ffffe00,$0ffffc00,$0ffff800,$0ffff000
        dc.l    $0fffe000,$00000000,$00000000,$00000000,$00000000,$00000000

Frame3:
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$003ff000
        dc.l    $003ff000,$003ff000,$003ff000,$003ff000,$003ff000,$003ff000
        dc.l    $003ff000,$003ff000,$003ff000,$003ff000,$003ff000,$003ff000
        dc.l    $003ff000,$003ff000,$003ff000,$003ff000,$003ff000,$003ff000
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00000000
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$007ff800
        dc.l    $00fffc00,$01fffe00,$03ffff00,$07ffff80,$0fffffc0,$1fffffe0
        dc.l    $3ffffff0,$7ffffff8,$fffffffc,$7ffffff8,$3ffffff0,$1fffffe0
        dc.l    $0fffffc0,$07ffff80,$03ffff00,$01fffe00,$00fffc00,$007ff800
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00000000

        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$007ff800
        dc.l    $00fffc00,$01fffe00,$03ffff00,$07ffff80,$0fffffc0,$1fffffe0
        dc.l    $3ffffff0,$7ffffff8,$fffffffc,$7ffffff8,$3ffffff0,$1fffffe0
        dc.l    $0fffffc0,$07ffff80,$03ffff00,$01fffe00,$00fffc00,$007ff800
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00000000

Frame4:
        dc.l    $00000000,$00000000,$00000000,$00000000,$00006000,$0000f000
        dc.l    $0001f800,$0003fc00,$0007fe00,$000fff00,$001fff80,$003fff00
        dc.l    $007ffe00,$00fffc00,$01fff800,$03fff000,$07ffe000,$0fffc000
        dc.l    $07ff8000,$03ff0000,$01fe0000,$00fc0000,$00780000,$00300000
        dc.l    $00000000,$00000000,$00000000,$00000000,$00000000,$00000000
        dc.l    $00000000,$00000000,$00000000,$00000000,$0fffe000,$0ffff000
        dc.l    $0ffff800,$0fffffc0,$0ffffe00,$0fffff00,$0fffff80,$0fffffc0
        dc.l    $0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0
        dc.l    $07ffffc0,$03ffffc0,$01ffffc0,$00ffffc0,$007fffc0,$003fffc0
        dc.l    $001fffc0,$00000000,$00000000,$00000000,$00000000,$00000000
```

```
        dc.l    $00000000,$00000000,$00000000,$00000000,$0fffe000,$0ffff000
        dc.l    $0ffff800,$0fffc00,$0fffe00,$0fffff00,$0fffff80,$0fffffc0
        dc.l    $0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0,$0fffffc0
        dc.l    $07ffffc0,$03ffffc0,$01ffffc0,$00ffffc0,$007fffc0,$003fffc0
        dc.l    $001fffc0,$00000000,$00000000,$00000000,$00000000,$00000000

;******************************************************************************

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 30, larghezza 3 words
; 2 bitplanes

Buffer:
        ds.w    30*3*2

; Il bitplane contiene un immagine da 1 plane 320*100
BITPLANE:

; plane 1
        ds.b    40*56                   ; 56 righe
        incbin  "sfondo320*100.raw"     ; 100 righe
        ds.b    40*100                  ; 100 righe

        ds.b    40*256                  ; plane 2

;******************************************************************************

        end
```

In questo esempio presentiamo una routine universale per disegnare bob.
La routine gestisce bob di dimensioni variabili. La posizione, le dimensioni
e gli indirizzi della figura e della maschera del bob sono passati come
parametri. Sulla base dei parametri vengono calcolati tutti i valori
da scrivere nei registri del blitter, usando formule viste in precedenza.
Conseguentemente, anche le routine di salvataggio e di ripristino dello
sfondo sono state modificate in modo da gestire rettangoli di dimensione
arbitraria. Fate attenzione che il Buffer di salvataggio usato da tali
routine sia grande abbastanza da contenere il rettangolo.
Usando queste routine e' possibile realizzare un bob animato in combinazione
con la routine di animazione vista nell'esempio lezione10l1.s (animazione
ciclica)
Notate che l'immagine di sfondo occupa solo parzialmente lo schermo, che
per il resto e' azzerato

Lezione10m2

```
; Lezione10m2.s Routine Universale Bob - versione INTERLEAVED
;               Tasto sinistro per uscire.

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

******************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
******************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

```
START:

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #2-1,D1         ; numero di bitplanes
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40,d0          ; + lunghezza riga
        addq.w  #8,a1
        dbra    d1,POINTBP

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:

; parametri per routine SalvaSfondo

        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  #32,d2                  ; dimensione X
        move.w  #30,d3                  ; dimensione Y
        bsr.w   SalvaSfondo             ; salva lo sfondo

; parametri per routine UniBob

        move.l  Frametab(pc),a0         ; mette il puntatore al fotogramma
                                        ; da disegnare in A0
        lea     2*4*30(a0),a1           ; puntatore alla maschera in A1
        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  #32,d2                  ; dimensione X
        move.w  #30,d3                  ; dimensione Y
        bsr.w   UniBob                  ; disegna il bob con la routine
                                        ; universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0
        BNE.S   Waity1

; parametri per routine RipristinaSfondo

        move.w  ogg_x(pc),d0            ; posizione X
        move.w  ogg_y(pc),d1            ; posizione Y
        move.w  #32,d2                  ; dimensione X
        move.w  #30,d3                  ; dimensione Y
        bsr.w   RipristinaSfondo        ; ripristina lo sfondo

        bsr.s   MuoviOggetto    ; sposta l'oggetto sullo schermo
```

```
        bsr.s   Animazione      ; sposta i fotogrammi nella tabella

        btst    #6,$bfe001      ; tasto sinistro del mouse premuto?
        bne.s   mouse           ; se no, torna a mouse
        rts


;***************************************************************************
; Questa routine muove il bob sullo schermo.
;***************************************************************************

MuoviOggetto:
        addq.w  #1,ogg_x        ; sposta in basso il bob
        cmp.w   #320-32,ogg_x   ; e' arrivato al bordo basso ?
        bls.s   EndMuovi        ; se no fine
        clr.w   ogg_x           ; altrimenti riparti dall'alto
EndMuovi
        rts

;***************************************************************************
; Questa routine crea l'animazione, spostando gli indirizzi dei fotogrammi
; in maniera che ogni volta il primo della tabella vada all'ultimo posto,
; mentra gli altri scorrono tutti di un posto in direzione del primo
;***************************************************************************

Animazione:
        addq.b  #1,ContaAnim    ; queste tre istruzioni fanno si' che il
        cmp.b   #4,ContaAnim    ; fotogramma venga cambiato una volta
        bne.s   NonCambiare     ; si e 3 no.
        clr.b   ContaAnim
        LEA     FRAMETAB(PC),a0 ; tabella dei fotogrammi
        MOVE.L  (a0),d0         ; salva il primo indirizzo in d0
        MOVE.L  4(a0),(a0)      ; sposta indietro gli altri indirizzi
        MOVE.L  4*2(a0),4(a0)   ; Queste istruzioni "ruotano" gli indirizzi
        MOVE.L  4*3(a0),4*2(a0) ; della tabella.
        MOVE.L  d0,4*3(a0)      ; metti l'ex primo indirizzo all'ottavo posto

NonCambiare:
        rts

ContaAnim:
        dc.w    0

; Questa e' la tabella degli indirizzi dei fotogrammi. Gli indirizzi
; presenti nella tabella vengono "ruotati" all'interno della tabella dalla
; routine Animazione, in modo che il primo nella lista sia la prima volta il
; fotogramma1, la volta dopo il Fotogramma2, poi il 3, il 4 e di nuovo il
; primo, ciclicamente. In questo modo basta prendere l'indirizzo che sta
; all'inizio della tabella ogni volta dopo il "rimescolamento" per avere gli
; indirizzi dei fotogrammi in sequenza.

FRAMETAB:
        DC.L    Frame1
        DC.L    Frame2
        DC.L    Frame3
        DC.L    Frame4

; variabili posizione BOB
OGG_Y:          dc.w    100     ; qui viene memorizzata la Y dell'oggetto
OGG_X:          dc.w    10      ; qui viene memorizzata la X dell'oggetto

;***************************************************************************
```

```
; Questa e' la routine universale per disegnare bob di forma e dimensioni
; arbitrarie. Tutti i parametri sono passati tramite registri.
; La routine funziona su schermo INTERLEAVED
;
; A0 - indirizzo figura bob
; A1 - indirizzo maschera bob
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;*****************************************************************************

;              ___  Oo          .:/
;             (___)o_o        ,,///;,   ,;/
;        //====--//(_)       o::::::::;;///
;                \\ ^       >:::::::::;;\\\
;                             ''\\\\\'" ';\

UniBob:

; calcolo indirizzo di partenza del blitter

        lea     bitplane,a2     ; indirizzo bitplane
        mulu.w  #2*40,d1        ; calcola indirizzo: ogni riga e' costituita da
                                ; 2 planes di 40 bytes ciascuno
        add.l   d1,a2           ; aggiungi ad indirizzo
        move.w  d0,d6           ; copia la X
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a2           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

        and.w   #$000f,d6       ; si selezionano i primi 4 bit della X perche'
                                ; vanno inseriti nello shifter dei canali A,B
        lsl.w   #8,d6           ; i 4 bit vengono spostati sul nibble alto
        lsl.w   #4,d6           ; della word. Questo e' il valore di BLTCON1

        move.w  d6,d5           ; copia per calcolare il valore di BLTCON0
        or.w    #$0FCA,d5       ; valori da mettere in BLTCON0

; calcolo modulo blitter

        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        addq.w  #2,d2           ; la blittata e' una word piu' larga
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d2,d4           ; modulo=larg. schermo-larg. rettangolo

; calcolo dimensione blittata

        mulu    #2,d3           ; moltiplica altezza per numero di planes
                                ; (per schermo interleaved)
                                ; in questo caso poiche' abbiamo 2 planes
                                ; si potrebbe usare la asl, ma in generale
                                ; (es. 3 planes) si deve usare la mulu

        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d2,d3           ; metti insieme le dimensioni

; inizializza i registri
```

```
        btst    #6,2(a5)
WBlit_u1:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit_u1

        move.l  #$ffff0000,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $0000 azzera l'ultima word

        move.w  d6,$42(a5)              ; BLTCON1 - valore di shift
                                        ; nessun modo speciale

        move.w  d5,$40(a5)              ; BLTCON0 - valore di shift
                                        ; cookie-cut

        move.l  #$fffefffe,$62(a5)      ; BLTBMOD e BLTAMOD=$fffe=-2 torna
                                        ; indietro all'inizio della riga.

        move.w  d4,$60(a5)              ; BLTCMOD valore calcolato
        move.w  d4,$66(a5)              ; BLTDMOD valore calcolato

        move.l  a1,$50(a5)              ; BLTAPT  (maschera)
        move.l  a2,$54(a5)              ; BLTDPT  (linee di schermo)
        move.l  a2,$48(a5)              ; BLTCPT  (linee di schermo)
        move.l  a0,$4c(a5)              ; BLTBPT  (figura bob)
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        rts

;*****************************************************************************
; Questa routine copia il rettangolo di sfondo che verra' sovrascritto dal
; BOB in un buffer. La routine gestisce un bob di dimensioni arbitrarie.
; Se usate questa routine per bob di dimensioni diverse, fate attenzione
; che il buffer possa contenere il bob di dimensione massima!
; La posizione e la dimensione del rettangolo sono dei parametri
;
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;*****************************************************************************

SalvaSfondo:
; calcolo indirizzo di partenza del blitter

        lea     bitplane,a1     ; indirizzo bitplane
        mulu.w  #2*40,d1        ; calcola indirizzo: ogni riga e' costituita da
                                ; 2 planes di 40 bytes ciascuno
        add.l   d1,a1           ; aggiungi ad indirizzo
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

; calcolo modulo blitter
        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        addq.w  #2,d2           ; la blittata e' larga 1 word in piu'
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d2,d4           ; modulo=larg. schermo-larg. rettangolo

; calcolo dimensione blittata
        mulu    #2,d3           ; moltiplica altezza per numero di planes
```

```
                                ; (per schermo interleaved)
                                ; in questo caso poiche' abbiamo 2 planes
                                ; si potrebbe usare la asl, ma in generale
                                ; (es. 3 planes) si deve usare la mulu
        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe' larghezza in words
        or      d2,d3           ; metti insieme le dimensioni

        lea     Buffer,a2       ; indirizzo destinazione

        btst    #6,2(a5) ; dmaconr
WBlit3:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.w  d4,$64(a5)              ; BLTAMOD valore calcolato
        move.w  #$0000,$66(a5)          ; BLTDMOD=0 nel buffer
        move.l  a1,$50(a5)              ; BLTAPT - ind. sorgente
        move.l  a2,$54(a5)              ; BLTDPT - buffer
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        rts

;*****************************************************************************
; Questa routine copia il contenuto del buffer nel rettangolo di schermo
; che lo conteneva prima del disegno del BOB. In questo modo viene anche
; cancellato il BOB dalla vecchia posizione. La routine gestisce un bob di
; dimensioni arbitrarie.
; Se usate questa routine per bob di dimensioni diverse, fate attenzione
; che il buffer possa contenere il bob di dimensione massima!
; La posizione e la dimensione del rettangolo sono dei parametri
;
; D0 - coordinata X del vertice superiore sinistro
; D1 - coordinata Y del vertice superiore sinistro
; D2 - larghezza rettangolo in pixel
; D3 - altezza rettangolo
;*****************************************************************************

RipristinaSfondo:
; calcolo indirizzo di partenza del blitter

        lea     bitplane,a1     ; indirizzo bitplane
        mulu.w  #2*40,d1        ; calcola indirizzo: ogni riga e' costituita da
                                ; 2 planes di 40 bytes ciascuno
        add.l   d1,a1           ; aggiungi ad indirizzo
        lsr.w   #3,d0           ; dividi per 8 la X
        and.w   #$fffe,d0       ; rendilo pari
        add.w   d0,a1           ; somma all'indirizzo del bitplane, trovando
                                ; l'indirizzo giusto di destinazione

; calcolo modulo blitter
        lsr.w   #3,d2           ; dividi per 8 la larghezza
        and.w   #$fffe,d2       ; azzerro il bit 0 (rendo pari)
        addq.w  #2,d2           ; la blittata e' larga 1 word in piu'
        move.w  #40,d4          ; larghezza schermo in bytes
        sub.w   d2,d4           ; modulo=larg. schermo-larg. rettangolo
```

```
; calcolo dimensione blittata
        mulu    #2,d3           ; moltiplica altezza per numero di planes
                                ; (per schermo interleaved)
                                ; in questo caso poiche‘ abbiamo 2 planes
                                ; si potrebbe usare la asl, ma in generale
                                ; (es. 3 planes) si deve usare la mulu
        lsl.w   #6,d3           ; altezza per 64
        lsr.w   #1,d2           ; larghezza in pixel diviso 16
                                ; cioe‘ larghezza in words
        or      d2,d3           ; metti insieme le dimensioni

        lea     Buffer,a2       ; indirizzo destinazione

        btst    #6,2(a5) ; dmaconr
WBlit4:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit4

        move.l  #$ffffffff,$44(a5)      ; BLTAFWM = $ffff fa passare tutto
                                        ; BLTALWM = $ffff fa passare tutto

        move.l  #$09f00000,$40(a5)      ; BLTCON0 e BLTCON1 copia da A a D
        move.w  d4,$66(a5)              ; BLTDMOD valore calcolato
        move.w  #$0000,$64(a5)          ; BLTAMOD=0 nel buffer
        move.l  a2,$50(a5)              ; BLTAPT - buffer
        move.l  a1,$54(a5)              ; BLTDPT - schermo
        move.w  d3,$58(a5)              ; BLTSIZE (via al blitter !)

        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,40         ; Bpl1Mod
        dc.w    $10a,40         ; Bpl2Mod

        dc.w    $100,$2200      ; bplcon0

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000

        dc.w    $180,$000       ; color0
        dc.w    $182,$00b       ; color1
        dc.w    $184,$cc0       ; color2
        dc.w    $186,$b00       ; color3

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************
; Questi sono i frames che compongono l’animazione

Frame1:
```

```
        dc.l    $00000000,$00020000
        dc.l    $00000000,$00070000
        dc.l    $00000000,$000f8000
        dc.l    $00000000,$001fc000
        dc.l    $00000000,$003fe000
        dc.l    $00000000,$007ff000
        dc.l    $00000000,$00fff800
        dc.l    $00000000,$01fffc00
        dc.l    $00000000,$03fffe00
        dc.l    $00000000,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $07ffff00,$07ffff00
        dc.l    $00000000,$07ffff00
        dc.l    $00000000,$03fffe00
        dc.l    $00000000,$01fffc00
        dc.l    $00000000,$00fff800
        dc.l    $00000000,$007ff000
        dc.l    $00000000,$003fe000
        dc.l    $00000000,$001fc000
        dc.l    $00000000,$000f8000
        dc.l    $00000000,$00070000
        dc.l    $00000000,$00020000

; maschera
        dc.l    $00020000
        dc.l    $00020000
        dc.l    $00070000
        dc.l    $00070000
        dc.l    $000f8000
        dc.l    $000f8000
        dc.l    $001fc000
        dc.l    $001fc000
        dc.l    $003fe000
        dc.l    $003fe000
        dc.l    $007ff000
        dc.l    $007ff000
        dc.l    $00fff800
        dc.l    $00fff800
        dc.l    $01fffc00
        dc.l    $01fffc00
        dc.l    $03fffe00
        dc.l    $03fffe00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
```

```
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $07ffff00
        dc.l    $03fffe00
        dc.l    $03fffe00
        dc.l    $01fffc00
        dc.l    $01fffc00
        dc.l    $00fff800
        dc.l    $00fff800
        dc.l    $007ff000
        dc.l    $007ff000
        dc.l    $003fe000
        dc.l    $003fe000
        dc.l    $001fc000
        dc.l    $001fc000
        dc.l    $000f8000
        dc.l    $000f8000
        dc.l    $00070000
        dc.l    $00070000
        dc.l    $00020000
        dc.l    $00020000

Frame2:
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$001fffc0
        dc.l    $00300000,$003fffc0
        dc.l    $00780000,$007fffc0
        dc.l    $00fc0000,$00ffffc0
        dc.l    $01fe0000,$01ffffc0
        dc.l    $03ff0000,$03ffffc0
        dc.l    $07ff8000,$07ffffc0
        dc.l    $0fffc000,$0fffffc0
        dc.l    $07ffe000,$0fffffc0
        dc.l    $03fff000,$0fffffc0
        dc.l    $01fff800,$0fffffc0
        dc.l    $00fffc00,$0fffffc0
        dc.l    $007ffe00,$0fffffc0
        dc.l    $003fff00,$0fffffc0
        dc.l    $001fff80,$0fffff80
        dc.l    $000fff00,$0fffff00
        dc.l    $0007fe00,$0ffffe00
        dc.l    $0003fc00,$0ffffc00
        dc.l    $0001f800,$0ffff800
        dc.l    $0000f000,$0ffff000
        dc.l    $00006000,$0fffe000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
```

```
        dc.l    $00000000,$00000000

        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $001fffc0
        dc.l    $001fffc0
        dc.l    $003fffc0
        dc.l    $003fffc0
        dc.l    $007fffc0
        dc.l    $007fffc0
        dc.l    $00ffffc0
        dc.l    $00ffffc0
        dc.l    $01ffffc0
        dc.l    $01ffffc0
        dc.l    $03ffffc0
        dc.l    $03ffffc0
        dc.l    $07ffffc0
        dc.l    $07ffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $0fffff80
        dc.l    $0fffff80
        dc.l    $0fffff00
        dc.l    $0fffff00
        dc.l    $0ffffe00
        dc.l    $0ffffe00
        dc.l    $0ffffc00
        dc.l    $0ffffc00
        dc.l    $0ffff800
        dc.l    $0ffff800
        dc.l    $0ffff000
        dc.l    $0ffff000
        dc.l    $0fffe000
        dc.l    $0fffe000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
```

```
Frame3:

        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $003ff000,$007ff800
        dc.l    $003ff000,$00fffc00
        dc.l    $003ff000,$01fffe00
        dc.l    $003ff000,$03ffff00
        dc.l    $003ff000,$07ffff80
        dc.l    $003ff000,$0fffffc0
        dc.l    $003ff000,$1fffffe0
        dc.l    $003ff000,$3ffffff0
        dc.l    $003ff000,$7ffffff8
        dc.l    $003ff000,$ffffffff
        dc.l    $003ff000,$7ffffff8
        dc.l    $003ff000,$3ffffff0
        dc.l    $003ff000,$1fffffe0
        dc.l    $003ff000,$0fffffc0
        dc.l    $003ff000,$07ffff80
        dc.l    $003ff000,$03ffff00
        dc.l    $003ff000,$01fffe00
        dc.l    $003ff000,$00fffc00
        dc.l    $003ff000,$007ff800
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000

        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $007ff800
        dc.l    $007ff800
        dc.l    $00fffc00
        dc.l    $00fffc00
        dc.l    $01fffe00
        dc.l    $01fffe00
        dc.l    $03ffff00
        dc.l    $03ffff00
        dc.l    $07ffff80
        dc.l    $07ffff80
        dc.l    $0fffffc0
        dc.l    $0fffffc0
        dc.l    $1fffffe0
        dc.l    $1fffffe0
        dc.l    $3ffffff0
        dc.l    $3ffffff0
        dc.l    $7ffffff8
        dc.l    $7ffffff8
```

```
        dc.l    $fffffffc
        dc.l    $fffffffc
        dc.l    $7fffffff8
        dc.l    $7fffffff8
        dc.l    $3fffffff0
        dc.l    $3fffffff0
        dc.l    $1ffffffe0
        dc.l    $1ffffffe0
        dc.l    $0ffffffc0
        dc.l    $0ffffffc0
        dc.l    $07ffff80
        dc.l    $07ffff80
        dc.l    $03ffff00
        dc.l    $03ffff00
        dc.l    $01fffe00
        dc.l    $01fffe00
        dc.l    $00fffc00
        dc.l    $00fffc00
        dc.l    $007ff800
        dc.l    $007ff800
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000

Frame4:

        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00006000,$0fffe000
        dc.l    $0000f000,$0ffff000
        dc.l    $0001f800,$0ffff800
        dc.l    $0003fc00,$0ffffc00
        dc.l    $0007fe00,$0fffffe00
        dc.l    $000fff00,$0fffff00
        dc.l    $001fff80,$0fffff80
        dc.l    $003fff00,$0fffffc0
        dc.l    $007ffe00,$0fffffc0
        dc.l    $00fffc00,$0fffffc0
        dc.l    $01fff800,$0fffffc0
        dc.l    $03fff000,$0fffffc0
        dc.l    $07ffe000,$0fffffc0
        dc.l    $0fffc000,$0fffffc0
        dc.l    $07ff8000,$07ffffc0
        dc.l    $03ff0000,$03ffffc0
        dc.l    $01fe0000,$01ffffc0
        dc.l    $00fc0000,$00ffffc0
        dc.l    $00780000,$007fffc0
        dc.l    $00300000,$003fffc0
        dc.l    $00000000,$001fffc0
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
```

```
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000
        dc.l    $00000000,$00000000

        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $0fffe000
        dc.l    $0fffe000
        dc.l    $0ffff000
        dc.l    $0ffff000
        dc.l    $0ffff800
        dc.l    $0ffff800
        dc.l    $0fffc00
        dc.l    $0fffc00
        dc.l    $0ffffe00
        dc.l    $0ffffe00
        dc.l    $0fffff00
        dc.l    $0fffff00
        dc.l    $0fffff80
        dc.l    $0fffff80
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $0ffffc0
        dc.l    $07fffc0
        dc.l    $07fffc0
        dc.l    $03fffc0
        dc.l    $03fffc0
        dc.l    $01fffc0
        dc.l    $01fffc0
        dc.l    $00fffc0
        dc.l    $00fffc0
        dc.l    $007fffc0
        dc.l    $007fffc0
        dc.l    $003fffc0
        dc.l    $003fffc0
        dc.l    $001fffc0
        dc.l    $001fffc0
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
        dc.l    $00000000
```

```
        dc.l    $00000000
        dc.l    $00000000

;***************************************************************************

        SECTION bitplane,BSS_C

; Questo e' il buffer nel quale salviamo di volta in volta lo sfondo.
; ha le stesse dimensioni di una blittata: altezza 30, larghezza 3 words
; 2 bitplanes

Buffer:
        ds.w    30*3*2

BITPLANE:

; 2 planes
        ds.b    40*256
        ds.b    40*256

;***************************************************************************

        end
```

In questo esempio mostriamo la versione rawblit della routine universale per disegnare i bob. Il programma e' identico a lezione10m1.s solo che viene impiegato uno schermo rawblit e di conseguenza, come sapete, cambiano un po' le formule per il calcolo di valori che vanno scritti nei registri. Anche se in questo caso non utilizziamo un'immagine di sfondo, le routine effettuano comunque il salvataggio e il ripristino. Potete disegnare voi un'immagine di sfondo (in versione rawblit) e mettercela senza modificare il sorgente!

Figura 26.9: Lezione 10m2

26.11 Lezione10n

```
; Lezione10n.s  Disegna una linea

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #100,d0         ; x1
        move.w  #100,d1         ; y1
        move.w  #220,d2         ; x2
        move.w  #120,d3         ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #160,d0         ; x1
        move.w  #85,d1          ; y1
        move.w  #160,d2         ; x2
        move.w  #140,d3         ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #80,d0          ; x1
        move.w  #130,d1         ; y1
        move.w  #50,d2          ; x2
        move.w  #190,d3         ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #$f0f0,d0       ; linea trattegiata
        bsr.w   SetPattern      ; definisce pattern
```

```
        move.w  #300,d0         ; x1
        move.w  #200,d1         ; y1
        move.w  #240,d2         ; x2
        move.w  #90,d3          ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #$4444,d0       ; linea trattegiata
        bsr.w   SetPattern      ; definisce pattern

        move.w  #210,d0         ; x1
        move.w  #24,d1          ; y1
        move.w  #68,d2          ; x2
        move.w  #50,d3          ; y2
        lea     bitplane,a0
        bsr.s   Drawline

mouse:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts


;*****************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

;           ("`-/")_.-'"``-._
;            . . `; -._    )-;-,_`)
;       FL  (v_,)'  _  )`-.\  ``-'
;          _.- _..-_/ / ((.'
;        ((,.-'   ((,/

Drawline:

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW2           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #0,d5           ; codice ottante
        bra.s   DRAWL
DRAW2:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW3           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
```

```
        moveq   #$18,d5         ; codice ottante
        bra.s   DRAWL
DRAW3:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$04,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW6           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.s   DRAWL
DRAW6:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW7           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$1c,d5         ; codice ottante
        bra.s   DRAWL
DRAW7:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$0c,d5         ; codice ottante

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante

DRAWL:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #$0002,d1       ; aggiunge la larghezza, pari a 2 words

        lsl.w   #$02,d3         ; D3=4*DY
        add.w   d2,d2           ; D2=2*DX

        btst    #6,2(a5)
WaitLine:
        btst    #6,2(a5)        ; aspetta blitter fermo
        bne     WaitLine
```

```
        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0001,d5       ; setta bit 0 (attiva line-mode)
        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;*****************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;*****************************************************************************

InitLine
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq.l #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;*****************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;*****************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern della linea!
        rts


;*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
```

```
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $180,$000       ; color0
        dc.w    $182,$eee       ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio presentiamo il tracciamento di linee. Esso viene realizzato mediante 3 diverse routines.
La routine "InitLine" setta i registri il cui contenuto e‘ indipendente dai parametri della linea (gli estremi) e che pertanto possono essere setati una volta sola all’inizio del programma.
La routine "SetPattern" definisce il pattern da usare per una linea. Ogni volta che si desidera usare un pattern diverso si deve eseguire questa routine. Al contrario, se ci sono diverse linee da tracciare con lo stesso pattern questa routine puo‘ essere eseguita una sola volta.
La routine "Drawline" e‘ la routine che disegna effetivamente la linea ed e‘ anche la piu‘ complessa. All’inizio vengono calcolati DX e DY e in base alle coordinate degli estremi, e viene determinato il codice ottante da usare.
Per compiere queste operazioni sono necessari una serie di sottrazioni e di confronti che prendono in esame tutti i possibili casi.
Successivamente vengono calcolati i valori da inserire negli altri registri, come spiegato nei commenti.
Notate che viene usato LF=$4A, il che provoca un EOR tra la linea e lo sfondo. Lo potete notare osservando le 2 linee che si intersecano: l’intersezione e‘ un pixel di valore 0. Se provate a porre LF=$CA noterete che l’intersezione e‘ invece un pixel di valore 1.

26.12 Lezione10o

```
; Lezione10o.s  Riempimento un poligono
;               alternare i tasti del mouse

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
```

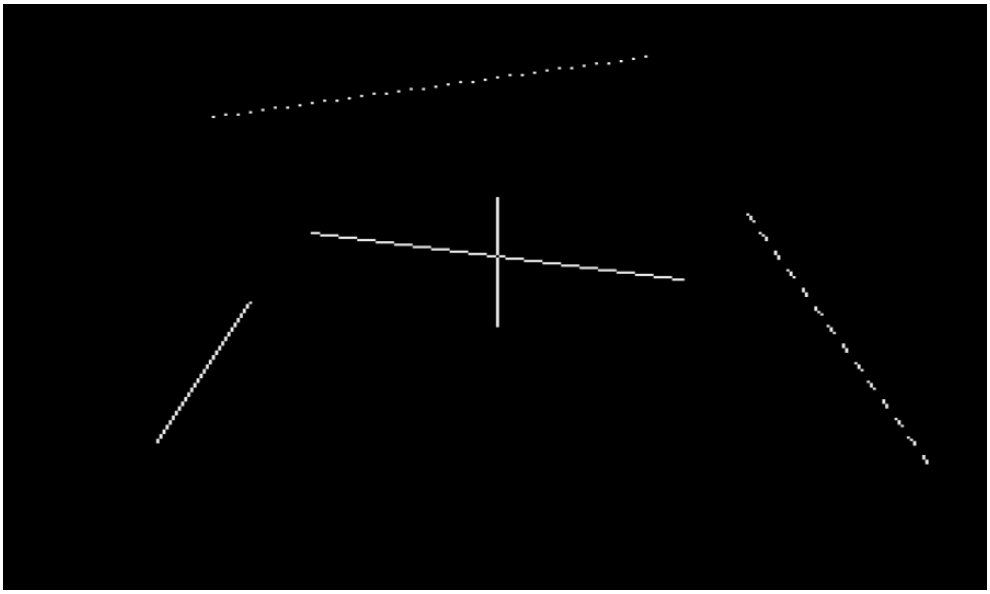


Figura 26.10: Lezione 10n

```
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #30,d0          ; x1
        move.w  #25,d1          ; y1
        move.w  #30,d2          ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
```

```
        bsr.w   Drawline

        move.w  #40,d0          ; x1
        move.w  #25,d1          ; y1
        move.w  #40,d2          ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.w   Drawline

        move.w  #45,d0          ; x1
        move.w  #25,d1          ; y1
        move.w  #55,d2          ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.w   Drawline

        move.w  #90,d0          ; x1
        move.w  #25,d1          ; y1
        move.w  #60,d2          ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.w   Drawline

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

        move.w  #0,d0                   ; inclusivo
        move.w  #0,d1                   ; CARRYIN = 0
        lea     bitplane+180*40+12,a0
        bsr.s   Fill

mouse2:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse2

        move.w  #$ffff,d0               ; esclusivo
        move.w  #0,d1                   ; CARRYIN = 0
        lea     bitplane+240*40+12,a0
        bsr.s   Fill

mouse3:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse3

        move.w  #0,d0                   ; inclusivo
        move.w  #$ffff,d1               ; CARRYIN = 1
        lea     bitplane+180*40+30,a0
        bsr.s   Fill

mouse4:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse4

        move.w  #$ffff,d0               ; esclusivo
        move.w  #$ffff,d1               ; CARRYIN = 1
        lea     bitplane+240*40+30,a0
        bsr.s   Fill

mouse:
        btst    #2,$dff016      ; mouse premuto?
        bne.s   mouse
```

```
        rts


;*****************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempmento. Il tipo di riempimento
; e‘ specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo destinazione
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;*****************************************************************************

;            ,-. __ .-,
;         --;‘. ’   ‘.’
;          / (  ^__^  )
;         ;   ‘(_‘’_)’ \
;         ’  ‘ .‘--’_,  ;
;       ~~‘-..._)))(((.’

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry

fill_esclusivo:
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2

fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #28,$64(a5)             ; BLTAMOD larghezza 6 words (40-12=28)
        move.w  #28,$66(a5)             ; BLTDMOD (40-12=28)

        move.l  #bitplane+80*40+12,$50(a5)
                                        ; BLTAPT (fisso al rettangolo sorgente)
                                        ; il rettangolo sorgente racchiude
                                        ; interamente le 2 linee.
                                        ; puntiamo l’ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*56)+6,$58(a5)      ; BLTSIZE (via al blitter !)
```

```
                                        ; larghezza 6 words
        rts                             ; altezza 56 righe (1 plane)


;***************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;***************************************************************************

Drawline:

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW2           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #0,d5           ; codice ottante
        bra.s   DRAWL
DRAW2:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW3           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$18,d5         ; codice ottante
        bra.s   DRAWL
DRAW3:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$04,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW6           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.s   DRAWL
DRAW6:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW7           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$1c,d5         ; codice ottante
        bra.s   DRAWL
```

```
DRAW7:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$0c,d5         ; codice ottante

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante

DRAWL:  mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #$0002,d1       ; aggiunge la larghezza, pari a 2 words

        lsl.w   #$02,d3         ; D3=4*DY
        add.w   d2,d2           ; D2=2*DX

        btst.b  #$06,$02(a5)
WaitLine:
        btst.b  #$06,$02(a5)    ; aspetta blitter fermo
        bne.s   WaitLine

        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0001,d5       ; setta bit 0 (attiva line-mode)
        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;****************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;****************************************************************************

InitLine
```

```
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************

SetPattern
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern della linea
        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

```
In questo esempio vediamo il funzionamento del fill-mode. Innanzitutto
tracciamo (con il blitter) delle linee sullo schermo. Poi copiamo il rettangolo
di schermo che le racchiude, effettuando il riempimento. Come potete osservare
le aree delimitate dalle linee vengono riempite.
La routine che si occupa del riempimento, legge il rettangolo sorgente da una
posizione fissa dello schermo (dove abbiamo tracciato inizialmente le linee)
mentre la destinazione e‘ variabile e viene passata attraverso un parametro.
Le dimensioni del rettangolo sono fisse e sono state calcolate in modo da
racchiudere tutte le linee tracciate.
La routine di Fill inoltre e‘ parametrizzata in modo da effettuare tutti i
tipi di fill: inclusivo, esclusivo, e con FILL_CARRYIN a 0 oppure a 1.
Osservando attentamente le immagini prodotte (sopratutto il rettangolo)
vedrete le differenze tra i tipi di fill.
```

Figura 26.11: Lezione 10o

26.13 Lezione10p

```
; Lezione10p.s  Riempimento un poligono con linee con pendenza < di 45 gradi
;               tasto destro per blittare, sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

```
START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #34,d0          ; x1
        move.w  #25,d1          ; y1
        move.w  #130,d2         ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.w   Drawline

        move.w  #220,d0         ; x1
        move.w  #25,d1          ; y1
        move.w  #140,d2         ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.s   Drawline

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

        move.w  #0,d0                   ; inclusivo
        move.w  #0,d1                   ; CARRYIN=0
        lea     bitplane+180*40+30,a0
        bsr.s   Fill

mouse2:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse2


        rts


;*****************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempmento. Il tipo di riempimento
; e‘ specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo destinazione
```

```
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;***************************************************************************

;         ("‘-’’-/").___..--’’"‘-._
;          ‘6_ 6  )   ‘-.  (     ).‘-.__.‘)
;          (_Y_.)’  ._   )  ‘._ ‘. ‘‘-..-’
;        _..‘--’_..-_/  /--’_.’ ,’
;       (il),-’’  (li),’  ((!.-’

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry

fill_esclusivo:
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2

fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #14,$64(a5)             ; BLTAMOD larghezza 13 words (40-26=14)
        move.w  #14,$66(a5)             ; BLTDMOD (40-26=14)

        move.l  #bitplane+80*40+28,$50(a5)
                                        ; BLTAPT (fisso al rettangolo sorgente)
                                        ; il rettangolo sorgente racchiude
                                        ; interamente le 2 linee.
                                        ; puntiamo l’ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*56)+13,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; larghezza 13 words
                                        ; altezza 56 righe (1 plane)
        rts

;***************************************************************************

;***************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l’indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
```

```
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

Drawline:

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW2           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$00,d5         ; codice ottante
        bra.s   DRAWL
DRAW2:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW3           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$18,d5         ; codice ottante
        bra.s   DRAWL
DRAW3:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$04,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW6           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.s   DRAWL
DRAW6:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW7           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$1c,d5         ; codice ottante
        bra.s   DRAWL
DRAW7:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$0c,d5         ; codice ottante

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante

DRAWL:
```

```
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #2,d1           ; aggiunge la larghezza, pari a 2 words

        lsl.w   #$02,d3         ; D3=4*DY
        add.w   d2,d2           ; D2=2*DX

        btst    #$06,$02(a5)
WaitLine:
        btst    #$06,$02(a5)    ; aspetta blitter fermo
        bne.s   WaitLine

        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0001,d5       ; setta bit 0 (attiva line-mode)
        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;***************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;***************************************************************************

InitLine
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
```

```
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************
SetPattern
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern della linea
        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio l'area da riempire e' delimitata da linee aventi pendenza minore di 45 gradi. Come abbiamo spiegato nella lezione, in queste condizioni il riempimento non viene effettuato correttamente.
Per risolvere questo problema bisogna impiegare un modo speciale di tracciamento delle linee, come vedremo nel prossimo esempio.

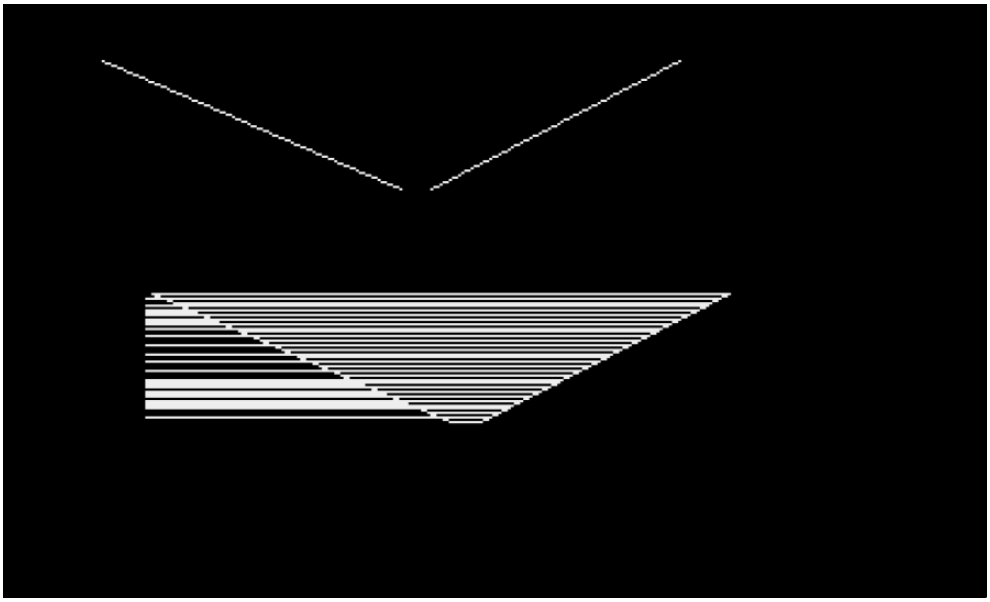


Figura 26.12: Lezione 10p

26.14 Lezione10q

```
; Lezione10q.s  Riempimento un poligono con linee con pendenza < di 45 gradi
;               tasto destro per blittare, sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
```

```
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #34,d0          ; x1
        move.w  #25,d1          ; y1
        move.w  #130,d2         ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #220,d0         ; x1
        move.w  #25,d1          ; y1
        move.w  #140,d2         ; x2
        move.w  #80,d3          ; y2
        lea     bitplane,a0
        bsr.s   DrawlineFill

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

        move.w  #0,d0                   ; inclusivo
        move.w  #0,d1                   ; CARRYIN=0
        lea     bitplane+180*40+30,a0
        bsr.s   Fill

mouse2:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse2
        rts


;*****************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempmento. Il tipo di riempimento
; e' specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo destinazione
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;*****************************************************************************

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry
```

```
fill_esclusivo:
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2

fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #14,$64(a5)             ; BLTAMOD larghezza 13 words (40-26=14)
        move.w  #14,$66(a5)             ; BLTDMOD (40-26=14)

        move.l  #bitplane+80*40+28,$50(a5)
                                        ; BLTAPT (fisso al rettangolo sorgente)
                                        ; il rettangolo sorgente racchiude
                                        ; interamente le 2 linee.
                                        ; puntiamo l'ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*56)+13,$58(a5)     ; BLTSIZE (via al blitter !)
                                        ; larghezza 13 words
                                        ; altezza 56 righe (1 plane)
        rts


;*****************************************************************************
; Questa routine effettua il disegno di una linea, usando la speciale modalita'
; che consente di effettuare correttamente il fill. Prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

;         ("'-''-/").___..--''"'-._
;          '6_ 6  )   '-.  (     ).'-.__.')
;          (_Y_.)'  ._   )  '._ '. ''-..-'
;        _..'--'_..-_/  /--'_.' ,'
;       (il),-''  (li),'  ((!.-'

DrawlineFill:

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW2           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
```

```
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$00,d5         ; codice ottante
        bra.s   DRAWL
DRAW2:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW3           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$18,d5         ; codice ottante
        bra.s   DRAWL
DRAW3:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$04,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW6           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.s   DRAWL
DRAW6:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW7           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$1c,d5         ; codice ottante
        bra.s   DRAWL
DRAW7:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$0c,d5         ; codice ottante

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante

DRAWL:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #2,d1           ; aggiunge la larghezza, pari a 2 words
```

```
        lsl.w   #$02,d3         ; D3=4*DY
        add.w   d2,d2           ; D2=2*DX

        btst    #$06,$02(a5)
WaitLine:
        btst    #$06,$02(a5)    ; aspetta blitter fermo
        bne.s   WaitLine

        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0003,d5       ; setta bit 0 (attiva line-mode), e
                                ; il bit 1 (linee speciali per fill)

        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;*****************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;*****************************************************************************

InitLine:
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;*****************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;*****************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern delle linee
```

```
        rts

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist


BITPLANE:
        dcb.b   40*256,0                ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio l'area da riempire e' delimitata da linee aventi pendenza
minore di 45 gradi. Impiegando la modalita' di disegno linee per il fill
riusciamo ad effettuare correttamente il riempimento.
Per disegnare le linee usiamo la routine "DrawLineFill" che e' identica
alla routine di disegno linee che abbiamo usato finora tranne per il fatto
che impiega questo modo speciale, settando l'apposito bit di BLTCON1.
Come potete vedere prima di effettuare il riempimento, questo modo speciale
produce linee che hanno un sempre un solo pixel per ogni linea orizzontale.
Se la pendenza delle linee fosse > di 45 gradi si avrebbero linee identiche
a quelle normali.
```

26.15 Lezione10r

```
; Lezione10r.s  Riempimento un poligono chiuso
;               tasto destro per blittare, sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
```

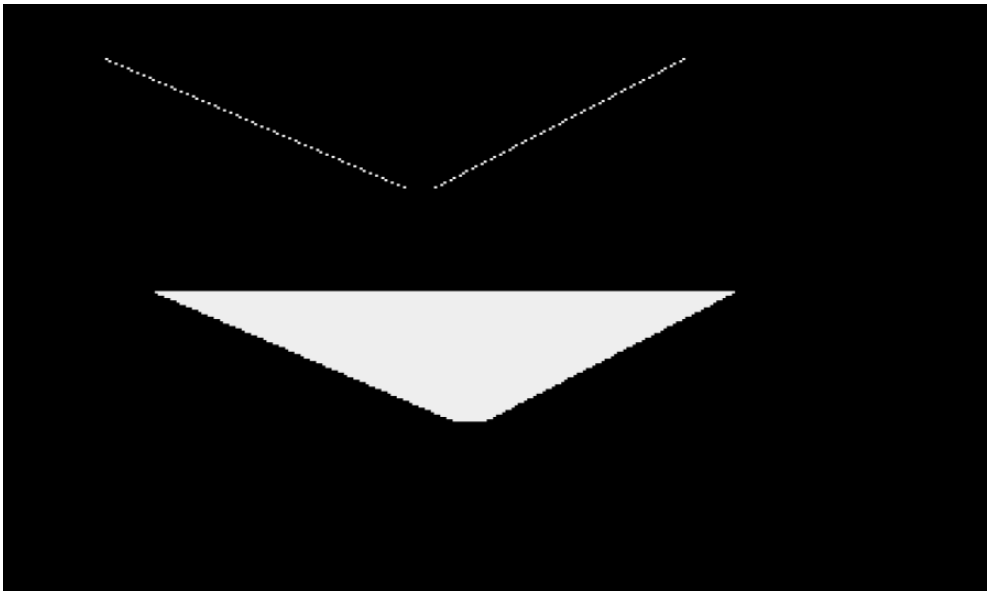


Figura 26.13: Lezione 10q

```
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #30,d0          ; x1
        move.w  #125,d1         ; y1
        move.w  #130,d2         ; x2
        move.w  #180,d3         ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #220,d0         ; x1
```

```
        move.w  #105,d1         ; y1
        move.w  #130,d2         ; x2
        move.w  #180,d3         ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #220,d0         ; x1
        move.w  #105,d1         ; y1
        move.w  #150,d2         ; x2
        move.w  #60,d3          ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #30,d0          ; x1
        move.w  #125,d1         ; y1
        move.w  #150,d2         ; x2
        move.w  #60,d3          ; y2
        lea     bitplane,a0
        bsr.s   DrawlineFill

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

        move.w  #0,d0                   ; inclusivo
        move.w  #0,d1                   ; CARRYIN=0
        lea     bitplane+180*40+28,a0
        bsr.s   Fill

mouse2:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse2
        rts


;*****************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempmento. Il tipo di riempimento
; e‘ specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo destinazione
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;*****************************************************************************

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry

fill_esclusivo:
```

```
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2

fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #12,$64(a5)             ; BLTAMOD larghezza 13 words (40-28=12)
        move.w  #12,$66(a5)             ; BLTDMOD (40-28=12)

        move.l  #bitplane+180*40+28,$50(a5)
                                        ; BLTAPT (fisso al rettangolo sorgente)
                                        ; il rettangolo sorgente racchiude
                                        ; interamente le 2 linee.
                                        ; puntiamo l'ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*121)+14,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; larghezza 14 words
                                        ; altezza 121 righe (1 plane)
        rts

;*****************************************************************************
; Questa routine effettua il disegno di una linea, usando la speciale modalita'
; che consente di effettuare correttamente il fill. Prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

;          ("'-/")_.-'"''-._
;           . . '; -._    )-;-,_')
;          (v_,)'  _  )'-.\  ''-'
;         _.- _..-_/ / ((.'
;       ((,.-'   ((,/

DrawlineFill:

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW2           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$00,d5         ; codice ottante
```

```
        bra.s   DRAWL
DRAW2:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW3           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$18,d5         ; codice ottante
        bra.s   DRAWL
DRAW3:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$04,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW6           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.s   DRAWL
DRAW6:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW7           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$1c,d5         ; codice ottante
        bra.s   DRAWL
DRAW7:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$0c,d5         ; codice ottante

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante

DRAWL:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #2,d1           ; aggiunge la larghezza, pari a 2 words

        lsl.w   #$02,d3         ; D3=4*DY
```

```
        add.w   d2,d2           ; D2=2*DX

        btst    #$06,$02(a5)
WaitLine:
        btst    #$06,$02(a5)    ; aspetta blitter fermo
        bne.s   WaitLine

        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0003,d5       ; setta bit 0 (attiva line-mode), e
                                ; il bit 1 (linee speciali per fill)

        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;***************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;***************************************************************************

InitLine
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************

SetPattern
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern delle linee
        rts
```

```
;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio disegnamo un poligono chiuso mediante linee.
Come potete vedere sorge un piccolo problema nei punti in cui le linee si
toccano, come abbiamo spiegato nella lezione.
Nel prossimo esempio vedrete la soluzione a questo problema.
```

26.16 Lezione10s

```
; Lezione10s.s  Riempimento corretto di un poligono chiuso
;               tasto destro per blittare, sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"
```

```
        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #30,d0          ; x1
        move.w  #125,d1         ; y1
        move.w  #130,d2         ; x2
        move.w  #180,d3         ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #220,d0         ; x1
        move.w  #105,d1         ; y1
        move.w  #130,d2         ; x2
        move.w  #180,d3         ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #220,d0         ; x1
        move.w  #105,d1         ; y1
        move.w  #150,d2         ; x2
        move.w  #60,d3          ; y2
        lea     bitplane,a0
        bsr.w   DrawlineFill

        move.w  #30,d0          ; x1
        move.w  #125,d1         ; y1
        move.w  #150,d2         ; x2
        move.w  #60,d3          ; y2
        lea     bitplane,a0
        bsr.s   DrawlineFill

mouse1:
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   mouse1

        move.w  #0,d0                   ; inclusivo
        move.w  #0,d1                   ; CARRYIN = 0
        lea     bitplane+180*40+28,a0
        bsr.s   Fill

mouse2:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse2


        rts
```

```
;***************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempmento. Il tipo di riempimento
; e‘ specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo destinazione
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;***************************************************************************

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry

fill_esclusivo:
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2

fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #12,$64(a5)             ; BLTAMOD larghezza 14 words (40-28=12)
        move.w  #12,$66(a5)             ; BLTDMOD (40-28=12)

        move.l  #bitplane+180*40+28,$50(a5)
                                        ; BLTAPT (fisso al rettangolo sorgente)
                                        ; il rettangolo sorgente racchiude
                                        ; interamente le 2 linee.
                                        ; puntiamo l’ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$54(a5)              ; BLTDPT  carica il parametro
        move.w  #(64*121)+14,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; larghezza 14 words
                                        ; altezza 121 righe (1 plane)
        rts


;***************************************************************************
; Questa routine effettua il disegno di una linea, usando la speciale modalita‘
; che consente di effettuare correttamente il fill. Inoltre la linea e‘
```

```
; disegnata dall'alto verso il basso e il primo bit viene lasciato immutato.
; Prende come parametri gli estremi della linea P1 e P2, e l'indirizzo del
; bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

;          ("‘-/")_.-’"‘‘-._
;           . . ‘; -._    )-;-,_‘)
;          (v_,)’  _  )‘-.\  ‘‘-’
;         _.- _..-_/ / ((.’
;       ((,.-’   ((,/

DrawlineFill:
        cmp.w   d1,d3           ; se D3>D1 i punti sono gia‘ nell’ordine giusto
        bge.w   Ordinati
        exg.l   d1,d3           ; altrimenti scambiali
        exg.l   d0,d2

Ordinati:
        sub.w   d1,d3           ; D3=DiffY  (e‘ sicuramente positivo)

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$00,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.w   DRAWL

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante

DRAWL:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
```

```
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        move.w  d0,d1           ; .. li copia in D1..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        move.l  a0,a1           ; copia l'indirizzo
        cmp.w   #7,d1           ; il numero del bit e' > 7 ?
        ble.s   NonCorreggi     ; se no salta

        addq.w  #1,a1           ; ..altrimenti punta al prossimo byte
        subq.w  #8,d1           ; e rendi il numero del bit < 7
NonCorreggi:
        not.b   d1              ; inverti la numerazione dei bit
                                ; questa istruzione e' necessaria perche'
                                ; all'interno del byte i bit sono numerati
                                ; da destra a sinistra, mentre le coordinate
                                ; dei pixel vanno da sinistra a destra
        bchg    d1,(a1)         ; inverti il primo pixel della linea

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #2,d1           ; aggiunge la larghezza, pari a 2 words

        lsl.w   #$02,d3         ; D3=4*DY
        add.w   d2,d2           ; D2=2*DX

        btst    #$06,$02(a5)
WaitLine:
        btst    #$06,$02(a5)    ; aspetta blitter fermo
        bne.s   WaitLine

        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0003,d5       ; setta bit 0 (attiva line-mode), e
                                ; il bit 1 (linee speciali per fill)

        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;***************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;***************************************************************************
```

```
InitLine:
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern della linea
        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

```
;*****************************************************************************

In questo esempio vediamo come risolvere il problema delle linee chiuse.
Come abbiamo spiegato nella lezione, modifichiamo la routine di tracciamento
linee in modo che essa disegni dall'alto in basso. Per questo all'inizio
vengono confrontate le coordinate Y dei 2 punti, e se necessario i 2 punti
vengono scambiati. In questo modo tra l'altro si semplifica il calcolo
dell'ottante in quanto siamo limitati a 4 sole possibilita'.
Inoltre il primo punto della linea viene invertito mediante una BCHG in
modo da annullare l'effetto del tracciamento. Per questo scopo sono necessari
un po' di calcoli in piu' al fine di determinare il numero del bit e il
giusto indirizzo. Infatti, a differenza del blitter, la BCHG opera un byte
alla volta, quindi se necessario bisogna puntare il byte basso della word
puntata dall'indirizzo. Inoltre bisogna anche invertire (con la NOT) la
numerazione dei bit perche' la BCHG numera i bit da destra a sinistra e
invece le coordinate sono numerate da sinistra a destra.
Le altre routines sono identiche.
```

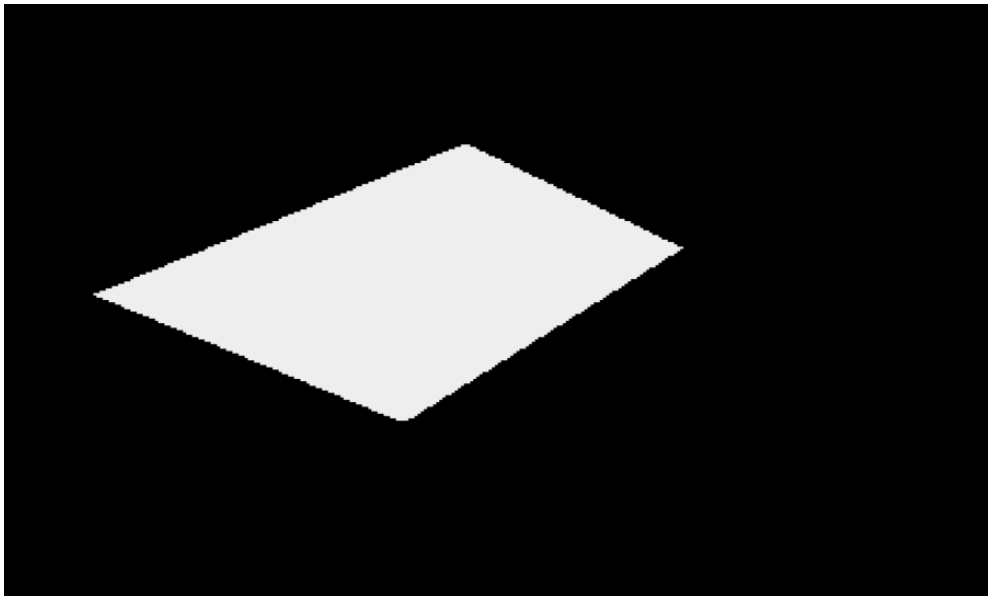


Figura 26.14: Lezione 10s

26.17 Lezione10t1

```
; Lezione10t1.s Disegna una linea larga 2 pixel

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************
```

```
                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #100,d0         ; x1
        move.w  #100,d1         ; y1
        move.w  #70,d2          ; x2
        move.w  #220,d3         ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #160,d0         ; x1
        move.w  #85,d1          ; y1
        move.w  #160,d2         ; x2
        move.w  #140,d3         ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #$f0f0,d0       ; linea trattegiata
        bsr.w   SetPattern      ; definisce pattern

        move.w  #300,d0         ; x1
        move.w  #200,d1         ; y1
        move.w  #240,d2         ; x2
        move.w  #90,d3          ; y2
        lea     bitplane,a0
        bsr.s   Drawline

mouse:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        rts


;******************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
```

```
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

Drawline:

* scelta ottante

        sub.w   d0,d2           ; D2=X2-X1
        bmi.s   DRAW4           ; se negativo salta, altrimenti D2=DiffX
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW2           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW1           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$10,d5         ; codice ottante
        bra.s   DRAWL
DRAW1:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$00,d5         ; codice ottante
        bra.s   DRAWL
DRAW2:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW3           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$18,d5         ; codice ottante
        bra.s   DRAWL
DRAW3:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$04,d5         ; codice ottante
        bra.s   DRAWL
DRAW4:
        neg.w   d2              ; rende D2 positivo
        sub.w   d1,d3           ; D3=Y2-Y1
        bmi.s   DRAW6           ; se negativo salta, altrimenti D3=DiffY
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW5           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$14,d5         ; codice ottante
        bra.s   DRAWL
DRAW5:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$08,d5         ; codice ottante
        bra.s   DRAWL
DRAW6:
        neg.w   d3              ; rende D3 positivo
        cmp.w   d3,d2           ; confronta DiffX e DiffY
        bmi.s   DRAW7           ; se D2<D3 salta..
                                ; .. altrimenti D3=DY e D2=DX
        moveq   #$1c,d5         ; codice ottante
        bra.s   DRAWL
DRAW7:
        exg.l   d2,d3           ; scambia D2 e D3, in modo che D3=DY e D2=DX
        moveq   #$0c,d5         ; codice ottante

; Quando l'esecuzione raggiunge questo punto, abbiamo:
; D2 = DX
; D3 = DY
; D5 = codice ottante
```

```
DRAWL:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0           ; aggiunge l'offset Y all'indirizzo

        move.w  d0,d1           ; copia la coordinata X
        and.w   #$000F,d0       ; seleziona i 4 bit piu' bassi della X..
        ror.w   #4,d0           ; .. e li sposta nei bit da 12 a 15
        or.w    #$0B4A,d0       ; con un OR ottengo il valore da scrivere
                                ; in BLTCON0. Con questo valore di LF ($4A)
                                ; si disegnano linee in EOR con lo sfondo.

        lsr.w   #4,d1           ; cancella i 4 bit bassi della X
        add.w   d1,d1           ; ottiene l'offset X in bytes
        add.w   d1,a0           ; aggiunge l'offset X all'indirizzo

        move.w  d2,d1           ; copia DX in D1
        addq.w  #1,d1           ; D1=DX+1
        lsl.w   #$06,d1         ; calcola in D1 il valore da mettere in BLTSIZE
        addq.w  #2,d1           ; aggiunge la larghezza, pari a 2 words

        lsl.w   #$02,d3         ; D3=4*DY
        add.w   d2,d2           ; D2=2*DX

        btst    #$06,$02(a5)
WaitLine:
        btst    #$06,$02(a5)    ; aspetta blitter fermo
        bne.s   WaitLine

        move.w  d3,$62(a5)      ; BLTBMOD=4*DY
        sub.w   d2,d3           ; D3=4*DY-2*DX
        move.w  d3,$52(a5)      ; BLTAPTL=4*DY-2*DX

                                ; prepara valore da scrivere in BLTCON1
        or.w    #$0001,d5       ; setta bit 0 (attiva line-mode)
        tst.w   d3
        bpl.s   OK1             ; se 4*DY-2*DX>0 salta..
        or.w    #$0040,d5       ; altrimenti setta il bit SIGN
OK1:
        move.w  d0,$40(a5)      ; BLTCON0
        move.w  d5,$42(a5)      ; BLTCON1
        sub.w   d2,d3           ; D3=4*DY-4*DX
        move.w  d3,$64(a5)      ; BLTAMOD=4*DY-4*DX
        move.l  a0,$48(a5)      ; BLTCPT - indirizzo schermo
        move.l  a0,$54(a5)      ; BLTDPT - indirizzo schermo
        move.w  d1,$58(a5)      ; BLTSIZE
        rts


;***************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra, settando BLTADAT in modo da tracciare
; linee doppie.
;***************************************************************************

InitLine:
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
```

```
;         )\._.,--....,''`.
;        /,   _.. \   _\  (`._ ,.
;       `._.-(,_..'--(,_..''`-.;.'

        move.w  #$C000,$74(a5)          ; BLTADAT = $C000 - linee doppie
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************
SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern linee
        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio vediamo come sia possibile tracciare linee larghe 2 pixel.
Si procede esattamente come per le linee normali ma si inizializza BLTADAT
con il valore $C000.

Lezione10t2

```
; Lezione10t2.s Routine tracciamento linee ottimizzata

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.w  #100,d0         ; x1
        move.w  #100,d1         ; y1
        move.w  #220,d2         ; x2
        move.w  #120,d3         ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #$f0f0,d0       ; linea trattegiata
        bsr.w   SetPattern      ; definisce pattern

        move.w  #300,d0         ; x1
        move.w  #200,d1         ; y1
        move.w  #240,d2         ; x2
        move.w  #90,d3          ; y2
        lea     bitplane,a0
        bsr.s   Drawline

        move.w  #$4444,d0       ; linea trattegiata
        bsr.w   SetPattern      ; definisce pattern

        move.w  #210,d0         ; x1
        move.w  #24,d1          ; y1
        move.w  #68,d2          ; x2
```

```
        move.w  #50,d3          ; y2
        lea     bitplane,a0
        bsr.s   Drawline

mouse:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts


;*****************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

;             .---.        .-----------
;            /     \  __  /    ------
;           / /     \(oo)/    -----
;          //////   ' \/ `   ---
;         //// / // :    : ---
;        // /   /  /`    '--
;       //          //..\\
;       -----------UU----UU-----
;                  '//||\\`
;                    ''``

; costanti

DL_Fill         =       0               ; 0=NOFILL / 1=FILL

        IFEQ    DL_Fill
DL_MInterns     =       $CA
        ELSE
DL_MInterns     =       $4A
        ENDC


DrawLine:
        sub.w   d1,d3   ; D3=Y2-Y1

        IFNE    DL_Fill
        beq.s   .end    ; per il fill non servono linee orizzontali
        ENDC

        bgt.s   .y2gy1  ; salta se positivo..
        exg     d0,d2   ; ..altrimenti scambia i punti
        add.w   d3,d1   ; mette in D1 la Y piu` piccola
        neg.w   d3      ; D3=DY
.y2gy1:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0
        moveq   #0,d1           ; D1 indice nella tabella ottanti
        sub.w   d0,d2           ; D2=X2-X1
        bge.s   .xdpos          ; salta se positivo..
        addq.w  #2,d1           ; ..altrimenti sposta l'indice
        neg.w   d2              ; e rendi positiva la differenza
.xdpos:
```

```
        moveq   #$f,d4          ; maschera per i 4 bit bassi
        and.w   d0,d4           ; selezionali in D4

        IFNE    DL_Fill         ; queste istruzioni vengono assemblate
                                ; solo se DL_Fill=1
        move.b  d4,d5           ; calcola numero del bit da invertire
        not.b   d5              ; (la BCHG numera i bit in modo inverso
        ENDC

        lsr.w   #3,d0           ; offset X:
                                ; Allinea a byte (serve per BCHG)
        add.w   d0,a0           ; aggiunge all'indirizzo
                                ; nota che anche se l'indirizzo
                                ; e' dispari non fa nulla perche'
                                ; il blitter non tiene conto del
                                ; bit meno significativo di BLTxPT

        ror.w   #4,d4           ; D4 = valore di shift A
        or.w    #$B00+DL_MInterns,d4    ; aggiunge l'opportuno
                                        ; Minterm (OR o EOR)
        swap    d4              ; valore di BLTCON0 nella word alta

        cmp.w   d2,d3           ; confronta DiffX e DiffY
        bge.s   .dygdx          ; salta se >=0..
        addq.w  #1,d1           ; altrimenti setta il bit 0 del'indice
        exg     d2,d3           ; e scambia le Diff
.dygdx:
        add.w   d2,d2           ; D2 = 2*DiffX
        move.w  d2,d0           ; copia in D0
        sub.w   d3,d0           ; D0 = 2*DiffX-DiffY
        addx.w  d1,d1           ; moltiplica per 2 l'indice e
                                ; contemporaneamente aggiunge il flag
                                ; X che vale 1 se 2*DiffX-DiffY<0
                                ; (settato dalla sub.w)
        move.b  Oktants(PC,d1.w),d4     ; legge l'ottante
        swap    d2                      ; valore BLTBMOD in word alta
        move.w  d0,d2                   ; word bassa D2=2*DiffX-DiffY
        sub.w   d3,d2                   ; word bassa D2=2*DiffX-2*DiffY
        moveq   #6,d1                   ; valore di shift e di test per
                                        ; la wait blitter

        lsl.w   d1,d3           ; calcola il valore di BLTSIZE
        add.w   #$42,d3

        lea     $52(a5),a1      ; A1 = indirizzo BLTAPTL
                                ; scrive alcuni registri
                                ; consecutivamente con delle
                                ; MOVE #XX,(Ax)+

        btst    d1,2(a5)        ; aspetta il blitter
.wb:
        btst    d1,2(a5)
        bne.s   .wb

        IFNE    DL_Fill         ; questa istruzione viene assemblata
                                ; solo se DL_Fill=1
        bchg    d5,(a0)         ; Inverte il primo bit della linea
        ENDC

        move.l  d4,$40(a5)      ; BLTCON0/1
        move.l  d2,$62(a5)      ; BLTBMOD e BLTAMOD
        move.l  a0,$48(a5)      ; BLTCPT
```

```
        move.w  d0,(a1)+        ; BLTAPTL
        move.l  a0,(a1)+        ; BLTDPT
        move.w  d3,(a1)         ; BLTSIZE
.end:
        rts

;
; se vogliamo eseguire linee per il fill, il codice ottante setta ad 1 il bit
; SING attraverso la costante SML

        IFNE    DL_Fill
SML             =       2
        ELSE
SML             =       0
        ENDC

; tabella ottanti

Oktants:
        dc.b    SML+1,SML+1+$40
        dc.b    SML+17,SML+17+$40
        dc.b    SML+9,SML+9+$40
        dc.b    SML+21,SML+21+$40

;***************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;***************************************************************************

InitLine
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern linee
        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C
```

```
COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio presentiamo una routine ottimizzata per il tracciamento di linee. La caratteristica principale di questa routine e' che i codici degli ottanti sono contenuti in una tabella. La routine a seconda delle posizioni dei punti calcola l'indice del giusto ottante nella tabella. Oltre a questo la routine impiega moltissime ottimizzazioni 68000. Questa routine contiene delle direttive assembler per l'assemblaggio condizionale. In base al valore della costante DL_Fill vengono assemblate o meno alcune parti della routine. In questo modo e' possibile riunire in un unico sorgente il codice sia per la versione normale che quello per la versione line-fill. Settando DL_Fill=0 si assembla la routine normale, mentre con DL_Fill=1 si assembla la versione per line fill. Per rendervene conto osservate (con il comando D di ASMONE) il codice prodotto nei 2 casi.

26.18 Lezione10u1

```
; Lezione10u1.s Linea in movimento
;               tasto sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

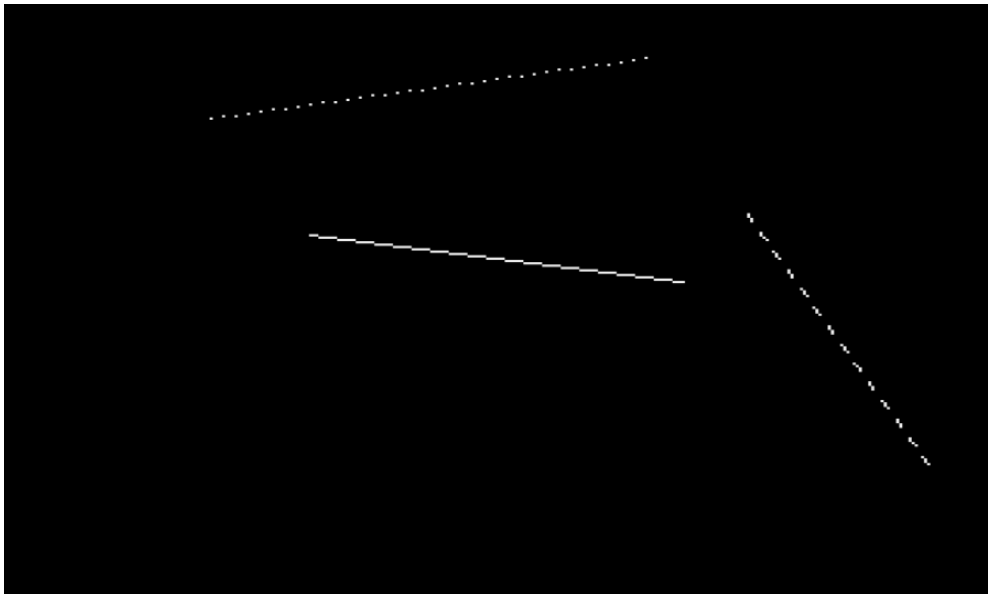


Figura 26.15: Lezione 10t2

```
START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
```

```
        bsr.w   CancellaSchermo ; pulisce lo schermo

        bsr.s   MuoviPunti      ; modifica le coordinate dei punti

; disegna la linea

        move.w  CoordX1(pc),d0  ; legge le coordinate dei punti
        move.w  CoordY1(pc),d1
        move.w  CoordX2(pc),d2
        move.w  CoordY2(pc),d3
        lea     bitplane,a0
        bsr.w   Drawline

        btst    #6,$bfe001      ; mouse premuto?
        bne.w   mouse
        rts

;*****************************************************************************
; Questa routine legge da tabelle le coordinate dei vari punti e le
; memorizza nelle apposite variabili.
; La lettura dalle tabelle viene effettuata mediante l'indirizzamento indiretto
; con indice. Per spostarci all'interno delle tabelle modifichiamo gli indici
; (che sono words) invece che i puntatori (longwords). Cio' ci permette di
; evitare di fuoriuscire dalla tabella con una semplice AND che mantiene
; l'indice compreso nell'intervallo 0 - 512 (infatti le tabelle sono composte
; da 256 valori words (512 bytes).
;*****************************************************************************

;         @..@
;        (----)
;       ( >__< )
;       ^^ ~~ ^^

MuoviPunti:
        lea     TabX(pc),a0

; coordinata X1

        move.w  indiceX1(pc),d0         ; indice della coordinata precedente
        add.w   addX1(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceX1             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordX1              ; copia la coordinata nella variabile

; coordinata X2

        move.w  indiceX2(pc),d0         ; indice della coordinata precedente
        add.w   addX2(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceX2             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordX2              ; copia la coordinata nella variabile

        lea     TabY(pc),a0

; coordinata Y1
```

```
        move.w  indiceY1(pc),d0         ; indice della coordinata precedente
        add.w   addY1(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceY1             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordY1              ; copia la coordinata nella variabile

; coordinata Y2

        move.w  indiceY2(pc),d0         ; indice della coordinata precedente
        add.w   addY2(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceY2             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordY2              ; copia la coordinata nella variabile

        rts

; questa tabella contiene le coordinate X

TabX:
        DC.W    $00A2,$00A6,$00A9,$00AD,$00B1,$00B4,$00B8,$00BB,$00BF,$00C3
        DC.W    $00C6,$00CA,$00CD,$00D1,$00D4,$00D8,$00DB,$00DE,$00E2,$00E5
        DC.W    $00E8,$00EC,$00EF,$00F2,$00F5,$00F8,$00FB,$00FE,$0101,$0103
        DC.W    $0106,$0109,$010B,$010E,$0110,$0113,$0115,$0117,$011A,$011C
        DC.W    $011E,$0120,$0122,$0123,$0125,$0127,$0128,$012A,$012B,$012D
        DC.W    $012E,$012F,$0130,$0131,$0132,$0133,$0133,$0134,$0135,$0135
        DC.W    $0135,$0136,$0136,$0136,$0136,$0136,$0136,$0135,$0135,$0135
        DC.W    $0134,$0133,$0133,$0132,$0131,$0130,$012F,$012E,$012D,$012B
        DC.W    $012A,$0128,$0127,$0125,$0123,$0122,$0120,$011E,$011C,$011A
        DC.W    $0117,$0115,$0113,$0110,$010E,$010B,$0109,$0106,$0103,$0101
        DC.W    $00FE,$00FB,$00F8,$00F5,$00F2,$00EF,$00EC,$00E8,$00E5,$00E2
        DC.W    $00DE,$00DB,$00D8,$00D4,$00D1,$00CD,$00CA,$00C6,$00C3,$00BF
        DC.W    $00BB,$00B8,$00B4,$00B1,$00AD,$00A9,$00A6,$00A2,$009E,$009A
        DC.W    $0097,$0093,$008F,$008C,$0088,$0085,$0081,$007D,$007A,$0076
        DC.W    $0073,$006F,$006C,$0068,$0065,$0062,$005E,$005B,$0058,$0054
        DC.W    $0051,$004E,$004B,$0048,$0045,$0042,$003F,$003D,$003A,$0037
        DC.W    $0035,$0032,$0030,$002D,$002B,$0029,$0026,$0024,$0022,$0020
        DC.W    $001E,$001D,$001B,$0019,$0018,$0016,$0015,$0013,$0012,$0011
        DC.W    $0010,$000F,$000E,$000D,$000D,$000C,$000B,$000B,$000B,$000A
        DC.W    $000A,$000A,$000A,$000A,$000A,$000B,$000B,$000B,$000C,$000D
        DC.W    $000D,$000E,$000F,$0010,$0011,$0012,$0013,$0015,$0016,$0018
        DC.W    $0019,$001B,$001D,$001E,$0020,$0022,$0024,$0026,$0029,$002B
        DC.W    $002D,$0030,$0032,$0035,$0037,$003A,$003D,$003F,$0042,$0045
        DC.W    $0048,$004B,$004E,$0051,$0054,$0058,$005B,$005E,$0062,$0065
        DC.W    $0068,$006C,$006F,$0073,$0076,$007A,$007D,$0081,$0085,$0088
        DC.W    $008C,$008F,$0093,$0097,$009A,$009E

; questa tabella contiene le coordinate Y

TabY:
        DC.W    $0080,$0083,$0086,$0088,$008B,$008E,$0090,$0093,$0096,$0098
        DC.W    $009B,$009E,$00A0,$00A3,$00A5,$00A8,$00AA,$00AD,$00AF,$00B2
        DC.W    $00B4,$00B6,$00B9,$00BB,$00BD,$00BF,$00C2,$00C4,$00C6,$00C8
        DC.W    $00CA,$00CC,$00CE,$00D0,$00D1,$00D3,$00D5,$00D7,$00D8,$00DA
        DC.W    $00DB,$00DD,$00DE,$00DF,$00E1,$00E2,$00E3,$00E4,$00E5,$00E6
        DC.W    $00E7,$00E8,$00E9,$00E9,$00EA,$00EB,$00EB,$00EC,$00EC,$00EC
        DC.W    $00ED,$00ED,$00ED,$00ED,$00ED,$00ED,$00ED,$00ED,$00EC,$00EC
```

```
        DC.W    $00EC,$00EB,$00EB,$00EA,$00E9,$00E9,$00E8,$00E7,$00E6,$00E5
        DC.W    $00E4,$00E3,$00E2,$00E1,$00DF,$00DE,$00DD,$00DB,$00DA,$00D8
        DC.W    $00D7,$00D5,$00D3,$00D1,$00D0,$00CE,$00CC,$00CA,$00C8,$00C6
        DC.W    $00C4,$00C2,$00BF,$00BD,$00BB,$00B9,$00B6,$00B4,$00B2,$00AF
        DC.W    $00AD,$00AA,$00A8,$00A5,$00A3,$00A0,$009E,$009B,$0098,$0096
        DC.W    $0093,$0090,$008E,$008B,$0088,$0086,$0083,$0080,$007E,$007B
        DC.W    $0078,$0076,$0073,$0070,$006E,$006B,$0068,$0066,$0063,$0060
        DC.W    $005E,$005B,$0059,$0056,$0054,$0051,$004F,$004C,$004A,$0048
        DC.W    $0045,$0043,$0041,$003F,$003C,$003A,$0038,$0036,$0034,$0032
        DC.W    $0030,$002E,$002D,$002B,$0029,$0027,$0026,$0024,$0023,$0021
        DC.W    $0020,$001F,$001D,$001C,$001B,$001A,$0019,$0018,$0017,$0016
        DC.W    $0015,$0015,$0014,$0013,$0013,$0012,$0012,$0012,$0011,$0011
        DC.W    $0011,$0011,$0011,$0011,$0011,$0011,$0012,$0012,$0012,$0013
        DC.W    $0013,$0014,$0015,$0015,$0016,$0017,$0018,$0019,$001A,$001B
        DC.W    $001C,$001D,$001F,$0020,$0021,$0023,$0024,$0026,$0027,$0029
        DC.W    $002B,$002D,$002E,$0030,$0032,$0034,$0036,$0038,$003A,$003C
        DC.W    $003F,$0041,$0043,$0045,$0048,$004A,$004C,$004F,$0051,$0054
        DC.W    $0056,$0059,$005B,$005E,$0060,$0063,$0066,$0068,$006B,$006E
        DC.W    $0070,$0073,$0076,$0078,$007B,$007E


; Qui sono memorizzate i volta in volta le coordinate dei vertici della linea

CoordX1:        dc.w    0
CoordY1:        dc.w    0
CoordX2:        dc.w    0
CoordY2:        dc.w    0

; Qui sono memorizzati per ogni coordinata gli indici all'interno della tabella

IndiceX1:       dc.w    10
IndiceY1:       dc.w    20
IndiceX2:       dc.w    30
IndiceY2:       dc.w    0

; Qui sono memorizzati per ogni coordinata i valori da aggiungere di volta
; in volta agli indici della tabella

addX1:  dc.w    -2
addY1:  dc.w    2
addX2:  dc.w    4
addY2:  dc.w    6

;******************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;******************************************************************************

; costanti

DL_Fill         =       0               ; 0=NOFILL / 1=FILL

        IFEQ    DL_Fill
DL_MInterns     =       $CA
        ELSE
DL_MInterns     =       $4A
        ENDC
```

```
DrawLine:
        sub.w   d1,d3   ; D3=Y2-Y1

        IFNE    DL_Fill
        beq.s   .end    ; per il fill non servono linee orizzontali
        ENDC

        bgt.s   .y2gy1  ; salta se positivo..
        exg     d0,d2   ; ..altrimenti scambia i punti
        add.w   d3,d1   ; mette in D1 la Y piu‘ piccola
        neg.w   d3      ; D3=DY
.y2gy1:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0
        moveq   #0,d1           ; D1 indice nella tabella ottanti
        sub.w   d0,d2           ; D2=X2-X1
        bge.s   .xdpos          ; salta se positivo..
        addq.w  #2,d1           ; ..altrimenti sposta l'indice
        neg.w   d2              ; e rendi positiva la differenza
.xdpos:
        moveq   #$f,d4          ; maschera per i 4 bit bassi
        and.w   d0,d4           ; selezionali in D4

        IFNE    DL_Fill         ; queste istruzioni vengono assemblate
                                ; solo se DL_Fill=1
        move.b  d4,d5           ; calcola numero del bit da invertire
        not.b   d5              ; (la BCHG numera i bit in modo inverso
        ENDC

        lsr.w   #3,d0           ; offset X:
                                ; Allinea a byte (serve per BCHG)
        add.w   d0,a0           ; aggiunge all'indirizzo
                                ; nota che anche se l'indirizzo
                                ; e‘ dispari non fa nulla perche‘
                                ; il blitter non tiene conto del
                                ; bit meno significativo di BLTxPT

        ror.w   #4,d4           ; D4 = valore di shift A
        or.w    #$B00+DL_MInterns,d4    ; aggiunge l'opportuno
                                        ; Minterm (OR o EOR)
        swap    d4              ; valore di BLTCON0 nella word alta

        cmp.w   d2,d3           ; confronta DiffX e DiffY
        bge.s   .dygdx          ; salta se >=0..
        addq.w  #1,d1           ; altrimenti setta il bit 0 del'indice
        exg     d2,d3           ; e scambia le Diff
.dygdx:
        add.w   d2,d2           ; D2 = 2*DiffX
        move.w  d2,d0           ; copia in D0
        sub.w   d3,d0           ; D0 = 2*DiffX-DiffY
        addx.w  d1,d1           ; moltiplica per 2 l'indice e
                                ; contemporaneamente aggiunge il flag
                                ; X che vale 1 se 2*DiffX-DiffY<0
                                ; (settato dalla sub.w)
        move.b  Oktants(PC,d1.w),d4     ; legge l'ottante
        swap    d2                      ; valore BLTBMOD in word alta
        move.w  d0,d2                   ; word bassa D2=2*DiffX-DiffY
        sub.w   d3,d2                   ; word bassa D2=2*DiffX-2*DiffY
        moveq   #6,d1                   ; valore di shift e di test per
                                        ; la wait blitter
```

```
        lsl.w   d1,d3           ; calcola il valore di BLTSIZE
        add.w   #$42,d3

        lea     $52(a5),a1      ; A1 = indirizzo BLTAPTL
                                ; scrive alcuni registri
                                ; consecutivamente con delle
                                ; MOVE #XX,(Ax)+

        btst    d1,2(a5)        ; aspetta il blitter
.wb:
        btst    d1,2(a5)
        bne.s   .wb

        IFNE    DL_Fill         ; questa istruzione viene assemblata
                                ; solo se DL_Fill=1
        bchg    d5,(a0)         ; Inverte il primo bit della linea
        ENDC

        move.l  d4,$40(a5)      ; BLTCON0/1
        move.l  d2,$62(a5)      ; BLTBMOD e BLTAMOD
        move.l  a0,$48(a5)      ; BLTCPT
        move.w  d0,(a1)+        ; BLTAPTL
        move.l  a0,(a1)+        ; BLTDPT
        move.w  d3,(a1)         ; BLTSIZE
.end:
        rts

;
; se vogliamo eseguire linee per il fill, il codice ottante setta ad 1 il bit
; SING attraverso la costante SML

        IFNE    DL_Fill
SML             =       2
        ELSE
SML             =       0
        ENDC

; tabella ottanti

Oktants:
        dc.b    SML+1,SML+1+$40
        dc.b    SML+17,SML+17+$40
        dc.b    SML+9,SML+9+$40
        dc.b    SML+21,SML+21+$40

;*****************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;*****************************************************************************

InitLine
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq.l #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
```

```
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern
        rts


;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
;***************************************************************************

CancellaSchermo:
        lea     bitplane,a0             ; indirizzo area da cancellare

        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move    #$0000,$66(a5)          ; BLTDMOD=0
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo
        rts


;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************
```

```
        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio vediamo come realizzare una linea che si muove sullo schermo.
La linea viene disegnata ad ogni frame in una differente posizione.
Le coordinate X e Y degli estremi della linea vengono lette da tabelle
sinusoidali.
```

Lezione10u2

```
; Lezione10u2.s Effetto con linee
;               tasto destro per vedere altri effetti, sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.w   InitLine        ; inizializza line-mode

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1

        bsr.w   ScambiaBuffer   ; questa routine scambia i 2 buffer

        bsr.w   CancellaSchermo ; pulisce lo schermo
```

```
        btst    #2,$dff016      ; tasto destro del mouse premuto?
        bne.s   NonCambia       ; se no salta..
        bsr.s   CambiaParametri ; ..altrimenti cambia i parametri delle linee
NonCambia:

        bsr.w   MuoviPunti      ; modifica le coordinate dei punti della prima
                                ; linea


        move.w  IndiceX1(pc),d4 ; legge gli indici della prima linea
        move.w  IndiceX2(pc),d5
        move.w  IndiceY1(pc),d6
        move.w  IndiceY2(pc),d7

        move.w  NumLines(pc),d0

LineLoop:

; disegna la linea

        movem.l d0-d7,-(a7)     ; salva i registri
        move.w  CoordX1(pc),d0  ; legge le coordinate dei punti
        move.w  CoordY1(pc),d1
        move.w  CoordX2(pc),d2
        move.w  CoordY2(pc),d3
        move.l  draw_buffer(pc),a0
        bsr.w   Drawline
        movem.l (a7)+,d0-d7     ; rileggi i registri

        bsr.w   NextLine

        dbra    d0,LineLoop     ; ripeti per ogni linea

        btst    #6,$bfe001      ; mouse premuto?
        bne.w   mouse

        rts


;***************************************************************************
; Questa routine cambia i valori dei parametri.
; I valori cambiati sono: "NumLines", i 4 "Indice", i 4 "Add", i 4 "NextAdd"
; I nuovi valori sono contenuti in un'apposita tabella.
; Siccome tutti i valori da cambiare sono consecutivi in memoria, si puo'
; utilizzare un unico loop di copia
;***************************************************************************

CambiaParametri:
        move.l  PointerParam(pc),a0     ; puntatore ai nuovi valori
        lea     NumLines(pc),a1         ; puntatore alle variabili

        moveq   #13-1,d0                ; numero di valori da cambiare
CambiaLoop:
        move.w  (a0)+,(a1)+             ; loop di copia
        dbra    d0,CambiaLoop

        cmp.l   #FineParam,a0           ; siamo alla fine della tabella?
        blo.s   NoRestart               ; se no salta..
        lea     TabParam(pc),a0         ; ..altrimenti ricomincia da capo

NoRestart:
        move.l  a0,PointerParam         ; memorizza il puntatore
```

```
; aspetta che venga rilasciato il pulsante del mouse

Waitmouse:
        btst    #2,$dff016              ; tasto destro del mouse premuto?
        beq.s   Waitmouse               ; se si aspetta

        rts

; puntatore alla tabella dei parametri

PointerParam:   dc.l    TabParam

; Tabella dei parametri
; potete provare a specificare voi dei parametri. I parametri (tranne il primo)
; DEVONO essere numeri PARI

TabParam:
        dc.w    $3a,0,$40,0,$40,2,2,2,2,8,8,$10,$10
        dc.w    $32,0,$80,0,$80,2,2,4,4,$7e,$80,$7e,$80

        dc.w    $3A,0,0,0,0,-2,2,4,4,$7e,$7e,$7e,$7e

        dc.w    $38,0,$68,0,$68,2,2,4,4,8,8,10,10
        dc.w    $28,$64,0,0,0,6,4,4,2,6,6,4,8
        dc.w    $3A,$40,$40,$40,$40,2,2,2,8,2,2,4,4
        dc.w    $39,2,0,$68,0,-2,2,4,4,8,8,10,10

        dc.w    $27,$64,0,0,0,8,4,4,2,4,2,2,4
        dc.w    $3A,0,$40,0,$40,2,2,4,4,4,4,$104,$104
FineParam:

;*****************************************************************************
; Questa routine legge da tabelle le coordinate dei vertici delle linee
; successiva alla prima e le memorizza nelle apposite variabili.
; La lettura dalle tabelle viene effettuata mediante l'indirizzamento indiretto
; con indice. Per spostarci all'interno delle tabelle modifichiamo gli indici
; (che sono words) invece che i puntatori (longwords). Cio' ci permette di
; evitare di fuoriuscire dalla tabella con una semplice AND che mantiene
; l'indice compreso nell'intervallo 0 - 512 (infatti le tabelle sono composte
; da 256 valori words (512 bytes).
; Gli indici delle coordinate precedenti sono memorizzati
; nei registri D4,D5,D6,D7
; I valori da sommare all'indice per passare da una linea all'altra sono
; memorizzari in apposite variabili.
;*****************************************************************************

NextLine:
        lea     TabX(pc),a0

; coordinata X1

        add.w   NextAddX1(pc),d4        ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d4                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  0(a0,d4.w),CoordX1      ; copia la coordinata dalla tabella
                                        ; nella variabile

; coordinata X2

        add.w   NextAddX2(pc),d5        ; modifica l'indice per puntare
```

```
                                        ; la nuova coordinata
        and.w   #$1FF,d5                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  0(a0,d5.w),CoordX2      ; copia la coordinata dalla tabella
                                        ; nella variabile


        lea     TabY(pc),a0

; coordinata Y1

        add.w   NextAddY1(pc),d6        ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d6                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  0(a0,d6.w),CoordY1      ; copia la coordinata dalla tabella
                                        ; nella variabile
; coordinata Y2

        add.w   NextAddY2(pc),d7        ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d7                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  0(a0,d7.w),CoordY2      ; copia la coordinata dalla tabella
                                        ; nella variabile
        rts

;***************************************************************************
; Questa routine legge da tabelle le coordinate dei vari punti e le
; memorizza nelle apposite variabili.
; La lettura dalle tabelle viene effettuata mediante l'indirizzamento indiretto
; con indice. Per spostarci all'interno delle tabelle modifichiamo gli indici
; (che sono words) invece che i puntatori (longwords). Cio' ci permette di
; evitare di fuoriuscire dalla tabella con una semplice AND che mantiene
; l'indice compreso nell'intervallo 0 - 512 (infatti le tabelle sono composte
; da 256 valori words (512 bytes).
;***************************************************************************

MuoviPunti:
        lea     TabX(pc),a0

; coordinata X1

        move.w  indiceX1(pc),d0         ; indice della coordinata precedente
        add.w   addX1(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceX1             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordX1              ; copia la coordinata nella variabile

; coordinata X2

        move.w  indiceX2(pc),d0         ; indice della coordinata precedente
        add.w   addX2(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceX2             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordX2              ; copia la coordinata nella variabile
```

```
        lea     TabY(pc),a0

; coordinata Y1

        move.w  indiceY1(pc),d0         ; indice della coordinata precedente
        add.w   addY1(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceY1             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordY1              ; copia la coordinata nella variabile

; coordinata Y2

        move.w  indiceY2(pc),d0         ; indice della coordinata precedente
        add.w   addY2(pc),d0            ; modifica l'indice per puntare
                                        ; la nuova coordinata
        and.w   #$1FF,d0                ; tiene l'indice all'interno della
                                        ; tabella
        move.w  d0,indiceY2             ; memorizza l'indice
        move.w  0(a0,d0.w),d1           ; legge la coordinata dalla tabella
        move.w  d1,CoordY2              ; copia la coordinata nella variabile
        rts

; questa tabella contiene le coordinate X

TabX:
        DC.W    $00A2,$00A6,$00A9,$00AD,$00B1,$00B4,$00B8,$00BB,$00BF,$00C3
        DC.W    $00C6,$00CA,$00CD,$00D1,$00D4,$00D8,$00DB,$00DE,$00E2,$00E5
        DC.W    $00E8,$00EC,$00EF,$00F2,$00F5,$00F8,$00FB,$00FE,$0101,$0103
        DC.W    $0106,$0109,$010B,$010E,$0110,$0113,$0115,$0117,$011A,$011C
        DC.W    $011E,$0120,$0122,$0123,$0125,$0127,$0128,$012A,$012B,$012D
        DC.W    $012E,$012F,$0130,$0131,$0132,$0133,$0133,$0134,$0135,$0135
        DC.W    $0135,$0136,$0136,$0136,$0136,$0136,$0136,$0135,$0135,$0135
        DC.W    $0134,$0133,$0133,$0132,$0131,$0130,$012F,$012E,$012D,$012B
        DC.W    $012A,$0128,$0127,$0125,$0123,$0122,$0120,$011E,$011C,$011A
        DC.W    $0117,$0115,$0113,$0110,$010E,$010B,$0109,$0106,$0103,$0101
        DC.W    $00FE,$00FB,$00F8,$00F5,$00F2,$00EF,$00EC,$00E8,$00E5,$00E2
        DC.W    $00DE,$00DB,$00D8,$00D4,$00D1,$00CD,$00CA,$00C6,$00C3,$00BF
        DC.W    $00BB,$00B8,$00B4,$00B1,$00AD,$00A9,$00A6,$00A2,$009E,$009A
        DC.W    $0097,$0093,$008F,$008C,$0088,$0085,$0081,$007D,$007A,$0076
        DC.W    $0073,$006F,$006C,$0068,$0065,$0062,$005E,$005B,$0058,$0054
        DC.W    $0051,$004E,$004B,$0048,$0045,$0042,$003F,$003D,$003A,$0037
        DC.W    $0035,$0032,$0030,$002D,$002B,$0029,$0026,$0024,$0022,$0020
        DC.W    $001E,$001D,$001B,$0019,$0018,$0016,$0015,$0013,$0012,$0011
        DC.W    $0010,$000F,$000E,$000D,$000D,$000C,$000B,$000B,$000B,$000A
        DC.W    $000A,$000A,$000A,$000A,$000A,$000B,$000B,$000B,$000C,$000D
        DC.W    $000D,$000E,$000F,$0010,$0011,$0012,$0013,$0015,$0016,$0018
        DC.W    $0019,$001B,$001D,$001E,$0020,$0022,$0024,$0026,$0029,$002B
        DC.W    $002D,$0030,$0032,$0035,$0037,$003A,$003D,$003F,$0042,$0045
        DC.W    $0048,$004B,$004E,$0051,$0054,$0058,$005B,$005E,$0062,$0065
        DC.W    $0068,$006C,$006F,$0073,$0076,$007A,$007D,$0081,$0085,$0088
        DC.W    $008C,$008F,$0093,$0097,$009A,$009E

; questa tabella contiene le coordinate Y

TabY:
        DC.W    $0080,$0083,$0086,$0088,$008B,$008E,$0090,$0093,$0096,$0098
        DC.W    $009B,$009E,$00A0,$00A3,$00A5,$00A8,$00AA,$00AD,$00AF,$00B2
        DC.W    $00B4,$00B6,$00B9,$00BB,$00BD,$00BF,$00C2,$00C4,$00C6,$00C8
```

```
        DC.W    $00CA,$00CC,$00CE,$00D0,$00D1,$00D3,$00D5,$00D7,$00D8,$00DA
        DC.W    $00DB,$00DD,$00DE,$00DF,$00E1,$00E2,$00E3,$00E4,$00E5,$00E6
        DC.W    $00E7,$00E8,$00E9,$00E9,$00EA,$00EB,$00EB,$00EC,$00EC,$00EC
        DC.W    $00ED,$00ED,$00ED,$00ED,$00ED,$00ED,$00ED,$00ED,$00EC,$00EC
        DC.W    $00EC,$00EB,$00EB,$00EA,$00E9,$00E9,$00E8,$00E7,$00E6,$00E5
        DC.W    $00E4,$00E3,$00E2,$00E1,$00DF,$00DE,$00DD,$00DB,$00DA,$00D8
        DC.W    $00D7,$00D5,$00D3,$00D1,$00D0,$00CE,$00CC,$00CA,$00C8,$00C6
        DC.W    $00C4,$00C2,$00BF,$00BD,$00BB,$00B9,$00B6,$00B4,$00B2,$00AF
        DC.W    $00AD,$00AA,$00A8,$00A5,$00A3,$00A0,$009E,$009B,$0098,$0096
        DC.W    $0093,$0090,$008E,$008B,$0088,$0086,$0083,$0080,$007E,$007B
        DC.W    $0078,$0076,$0073,$0070,$006E,$006B,$0068,$0066,$0063,$0060
        DC.W    $005E,$005B,$0059,$0056,$0054,$0051,$004F,$004C,$004A,$0048
        DC.W    $0045,$0043,$0041,$003F,$003C,$003A,$0038,$0036,$0034,$0032
        DC.W    $0030,$002E,$002D,$002B,$0029,$0027,$0026,$0024,$0023,$0021
        DC.W    $0020,$001F,$001D,$001C,$001B,$001A,$0019,$0018,$0017,$0016
        DC.W    $0015,$0015,$0014,$0013,$0013,$0012,$0012,$0012,$0011,$0011
        DC.W    $0011,$0011,$0011,$0011,$0011,$0011,$0012,$0012,$0012,$0013
        DC.W    $0013,$0014,$0015,$0015,$0016,$0017,$0018,$0019,$001A,$001B
        DC.W    $001C,$001D,$001F,$0020,$0021,$0023,$0024,$0026,$0027,$0029
        DC.W    $002B,$002D,$002E,$0030,$0032,$0034,$0036,$0038,$003A,$003C
        DC.W    $003F,$0041,$0043,$0045,$0048,$004A,$004C,$004F,$0051,$0054
        DC.W    $0056,$0059,$005B,$005E,$0060,$0063,$0066,$0068,$006B,$006E
        DC.W    $0070,$0073,$0076,$0078,$007B,$007E


; Qui sono memorizzate i volta in volta le coordinate dei vertici della linea

CoordX1:        dc.w    0
CoordY1:        dc.w    0
CoordX2:        dc.w    0
CoordY2:        dc.w    0

; Qui e‘ memorizzato il numero di linee disegnate

NumLines:       dc.w    10

; Qui sono memorizzati per ogni coordinata gli indici all’interno della tabella

IndiceX1:       dc.w    20
IndiceY1:       dc.w    50
IndiceX2:       dc.w    30
IndiceY2:       dc.w    40

; Qui sono memorizzati per ogni coordinata i valori da aggiungere ad ogni frame
; agli indici della tabella per i vertici della prima linea

addX1:  dc.w    4
addY1:  dc.w    -6
addX2:  dc.w    -2
addY2:  dc.w    2

; Qui sono memorizzati per ogni coordinata i valori da aggiungere agli indici
; della tabella per i vertici delle linee successive

NextaddX1:      dc.w    10
NextaddY1:      dc.w    14
NextaddX2:      dc.w    6
NextaddY2:      dc.w    -4

;******************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l’indirizzo del bitplane su cui disegnarla.
```

```
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

; costanti

DL_Fill         =       0               ; 0=NOFILL / 1=FILL

        IFEQ    DL_Fill
DL_MInterns     =       $CA
        ELSE
DL_MInterns     =       $4A
        ENDC


DrawLine:
        sub.w   d1,d3   ; D3=Y2-Y1

        IFNE    DL_Fill
        beq.s   .end    ; per il fill non servono linee orizzontali
        ENDC

        bgt.s   .y2gy1  ; salta se positivo..
        exg     d0,d2   ; ..altrimenti scambia i punti
        add.w   d3,d1   ; mette in D1 la Y piu' piccola
        neg.w   d3      ; D3=DY
.y2gy1:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0
        moveq   #0,d1           ; D1 indice nella tabella ottanti
        sub.w   d0,d2           ; D2=X2-X1
        bge.s   .xdpos          ; salta se positivo..
        addq.w  #2,d1           ; ..altrimenti sposta l'indice
        neg.w   d2              ; e rendi positiva la differenza
.xdpos:
        moveq   #$f,d4          ; maschera per i 4 bit bassi
        and.w   d0,d4           ; selezionali in D4

        IFNE    DL_Fill         ; queste istruzioni vengono assemblate
                                ; solo se DL_Fill=1
        move.b  d4,d5           ; calcola numero del bit da invertire
        not.b   d5              ; (la BCHG numera i bit in modo inverso
        ENDC

        lsr.w   #3,d0           ; offset X:
                                ; Allinea a byte (serve per BCHG)
        add.w   d0,a0           ; aggiunge all'indirizzo
                                ; nota che anche se l'indirizzo
                                ; e' dispari non fa nulla perche'
                                ; il blitter non tiene conto del
                                ; bit meno significativo di BLTxPT

        ror.w   #4,d4           ; D4 = valore di shift A
        or.w    #$B00+DL_MInterns,d4    ; aggiunge l'opportuno
                                        ; Minterm (OR o EOR)
        swap    d4              ; valore di BLTCON0 nella word alta

        cmp.w   d2,d3           ; confronta DiffX e DiffY
        bge.s   .dygdx          ; salta se >=0..
```

```
        addq.w  #1,d1           ; altrimenti setta il bit 0 del'indice
        exg     d2,d3           ; e scambia le Diff
.dygdx:
        add.w   d2,d2           ; D2 = 2*DiffX
        move.w  d2,d0           ; copia in D0
        sub.w   d3,d0           ; D0 = 2*DiffX-DiffY
        addx.w  d1,d1           ; moltiplica per 2 l'indice e
                                ; contemporaneamente aggiunge il flag
                                ; X che vale 1 se 2*DiffX-DiffY<0
                                ; (settato dalla sub.w)
        move.b  Oktants(PC,d1.w),d4     ; legge l'ottante
        swap    d2                      ; valore BLTBMOD in word alta
        move.w  d0,d2                   ; word bassa D2=2*DiffX-DiffY
        sub.w   d3,d2                   ; word bassa D2=2*DiffX-2*DiffY
        moveq   #6,d1                   ; valore di shift e di test per
                                        ; la wait blitter

        lsl.w   d1,d3           ; calcola il valore di BLTSIZE
        add.w   #$42,d3

        lea     $52(a5),a1      ; A1 = indirizzo BLTAPTL
                                ; scrive alcuni registri
                                ; consecutivamente con delle
                                ; MOVE #XX,(Ax)+

        btst    d1,2(a5)        ; aspetta il blitter
.wb:
        btst    d1,2(a5)
        bne.s   .wb

        IFNE    DL_Fill         ; questa istruzione viene assemblata
                                ; solo se DL_Fill=1
        bchg    d5,(a0)         ; Inverte il primo bit della linea
        ENDC

        move.l  d4,$40(a5)      ; BLTCON0/1
        move.l  d2,$62(a5)      ; BLTBMOD e BLTAMOD
        move.l  a0,$48(a5)      ; BLTCPT
        move.w  d0,(a1)+        ; BLTAPTL
        move.l  a0,(a1)+        ; BLTDPT
        move.w  d3,(a1)         ; BLTSIZE
.end:
        rts

;
; se vogliamo eseguire linee per il fill, il codice ottante setta ad 1 il bit
; SING attraverso la costante SML

        IFNE    DL_Fill
SML             =       2
        ELSE
SML             =       0
        ENDC

; tabella ottanti

Oktants:
        dc.b    SML+1,SML+1+$40
        dc.b    SML+17,SML+17+$40
        dc.b    SML+9,SML+9+$40
        dc.b    SML+21,SML+21+$40
```

```
;***************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;***************************************************************************

InitLine
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;***************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;***************************************************************************
SetPattern
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern della linea
        rts


;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
;***************************************************************************

CancellaSchermo:
        move.l  draw_buffer(pc),a0      ; indirizzo area da cancellare

        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0000,$66(a5)          ; BLTDMOD=0
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo
        rts

;***************************************************************************
; Questa routine scambia i 2 buffer scambiando gli indirizzi nelle
; variabili VIEW_BUFFER e  DRAW_BUFFER.
; Inoltre aggiorna nella copperlist le istruzioni che caricano i registri
; BPLxPT, in modo che puntino al nuovo buffer da visualizzare.
;***************************************************************************

ScambiaBuffer
        move.l  draw_buffer(pc),d0              ; scambia il contenuto
        move.l  view_buffer(pc),draw_buffer     ; delle variabili
```

```
        move.l  d0,view_buffer                  ; in d0 c'e' l'indirizzo
                                                ; del nuovo buffer
                                                ; da visualizzare

; aggiorna la copperlist puntando i bitplanes del nuovo buffer da visualizzare

        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        rts


; puntatori ai 2 buffer

view_buffer:    dc.l    BITPLANE        ; buffer visualizzato
draw_buffer:    dc.l    BITPLANEb       ; buffer di disegno

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

; buffer 1

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

; buffer 2

BITPLANEb:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************
```

In questo esempio aggiungiamo altre linee al programma lezione10h1.s
realizzando effetti piu' complessi. Le linee successive alla prima vengono
disegnate prendendo come coordinate dei valori letti sempre dalla tabella.

```
Per calcolare le nuove coordinate vengono aggiunti agli indici dei valori
ad ogni linea. Tali valori sono contenuti nelle variabili "NextAdd".
Inoltre premendo il bottone destro si possono variare tutti i valori
delle variabili. In questo modo si possono ottenere nuovi effetti semplicemente
variando i parametri.
```

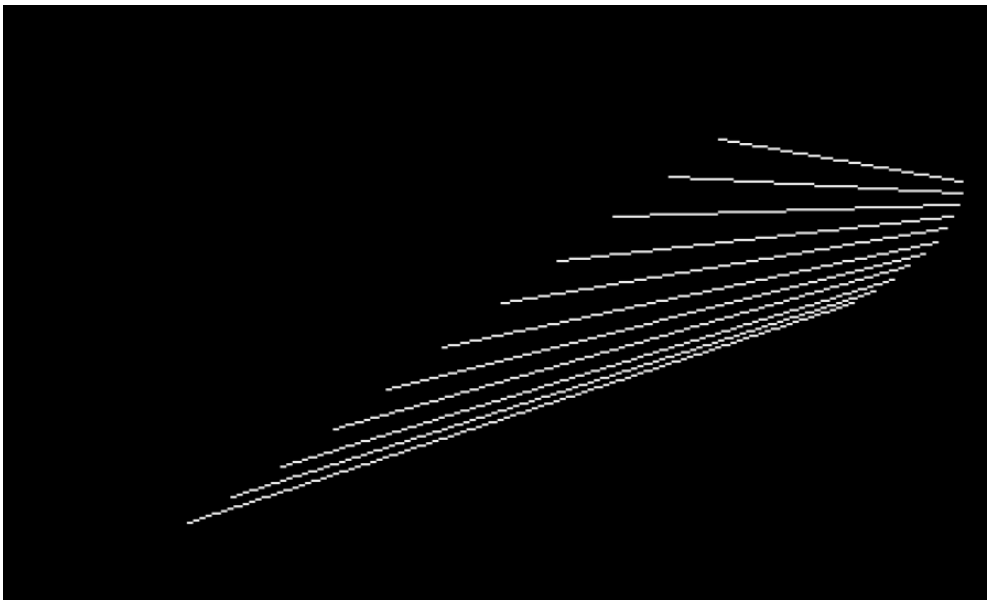


Figura 26.16: Lezione 10u2

26.19 Lezione10v

```
; Lezione10v.s  Poligono che ruota.
;               tasto sinistro per uscire

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA


START:
;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
```

```
        move.w  d0,2(a1)

        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1

        bsr.w   CancellaSchermo ; pulisce lo schermo

        bsr.w   MuoviPunti      ; modifica le coordinate dei punti

        bsr.w   InitLine        ; inizializza line-mode

; disegna la linea tra i punti 1 e 2

        move.w  Point1(pc),d0
        move.w  Point1+2(pc),d1
        move.w  Point2(pc),d2
        move.w  Point2+2(pc),d3
        lea     bitplane,a0
        bsr.w   Drawline

; disegna la linea tra i punti 2 e 3

        move.w  Point2(pc),d0
        move.w  Point2+2(pc),d1
        move.w  Point3(pc),d2
        move.w  Point3+2(pc),d3
        lea     bitplane,a0
        bsr.w   Drawline

; disegna la linea tra i punti 3 e 4

        move.w  Point3(pc),d0
        move.w  Point3+2(pc),d1
        move.w  Point4(pc),d2
        move.w  Point4+2(pc),d3
        lea     bitplane,a0
        bsr.w   Drawline

; disegna la linea tra i punti 4 e 1

        move.w  Point4(pc),d0
        move.w  Point4+2(pc),d1
        move.w  Point1(pc),d2
        move.w  Point1+2(pc),d3
        lea     bitplane,a0
```

```
        bsr.w   Drawline

        moveq   #0,d0
        moveq   #0,d1
        lea     bitplane+178*40-2,a0
        bsr.w   Fill

        btst    #6,$bfe001      ; mouse premuto?
        bne.w   mouse
        rts

;***************************************************************************
; Questa routine legge da una tabella le coordinate dei vari punti e le
; memorizza nelle apposite variabili.
;***************************************************************************

MuoviPunti:
        ADDQ.L  #2,TABXPOINT            ; Fai puntare alla word successiva
        MOVE.L  TABXPOINT(PC),A0        ; indirizzo contenuto in long TABXPOINT
                                        ; copiato in a0
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BNE.S   NOBSTARTX               ; non ancora? allora continua
        MOVE.L  #TABX-2,TABXPOINT       ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        MOVE.W  (A0),Point1             ; copia il valore della coordinata
                                        ; del punto 1 nella variabile apposita

        LEA     50(A0),A0               ; Coordinata del punto seguente
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BLE.S   NOBSTARTX2              ; no allora leggi
        SUB.L   #FINETABX-TABX,A0       ; altrimenti torna indietro nella
                                        ; tabella
NOBSTARTX2:
        MOVE.W  (A0),Point2             ; copia il valore della coordinata
                                        ; del punto 2 nella variabile apposita

        LEA     50(A0),A0               ; Coordinata del punto seguente
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BLE.S   NOBSTARTX3              ; no allora leggi
        SUB.L   #FINETABX-TABX,A0       ; altrimenti torna indietro nella
                                        ; tabella
NOBSTARTX3:
        MOVE.W  (A0),Point3             ; copia il valore della coordinata
                                        ; del punto 3 nella variabile apposita

        LEA     50(A0),A0               ; Coordinata del punto seguente
        CMP.L   #FINETABX-2,A0          ; Siamo all'ultima word della TAB?
        BLE.S   NOBSTARTX4              ; no allora leggi
        SUB.L   #FINETABX-TABX,A0       ; altrimenti torna indietro nella
                                        ; tabella
NOBSTARTX4:
        MOVE.W  (A0),Point4             ; copia il valore della coordinata
                                        ; del punto 4 nella variabile apposita

        ADDQ.L  #2,TABYPOINT            ; Fai puntare alla word successiva
        MOVE.L  TABYPOINT(PC),A0        ; indirizzo contenuto in long TABYPOINT
                                        ; copiato in a0
        CMP.L   #FINETABY-2,A0          ; Siamo all'ultima word della TAB?
        BNE.S   NOBSTARTY               ; non ancora? allora continua
        MOVE.L  #TABY-2,TABYPOINT       ; Riparti a puntare dalla prima word-2
NOBSTARTY:
        MOVE.W  (A0),Point1+2           ; copia il valore della coordinata
```

```
                                    ; del punto 1 nella variabile apposita

        LEA     50(A0),A0           ; Coordinata del punto seguente
        CMP.L   #FINETABY-2,A0      ; Siamo all'ultima word della TAB?
        BLE.S   NOBSTARTY2          ; no allora leggi
        SUB.L   #FINETABY-TABY,A0   ; altrimenti torna indietro nella
                                    ; tabella
NOBSTARTY2:
        MOVE.W  (A0),Point2+2       ; copia il valore della coordinata
                                    ; del punto 2 nella variabile apposita

        LEA     50(A0),A0           ; Coordinata del punto seguente
        CMP.L   #FINETABY-2,A0      ; Siamo all'ultima word della TAB?
        BLE.S   NOBSTARTY3          ; no allora leggi
        SUB.L   #FINETABY-TABY,A0   ; altrimenti torna indietro nella
                                    ; tabella
NOBSTARTY3:
        MOVE.W  (A0),Point3+2       ; copia il valore della coordinata
                                    ; del punto 3 nella variabile apposita

        LEA     50(A0),A0           ; Coordinata del punto seguente
        CMP.L   #FINETABY-2,A0      ; Siamo all'ultima word della TAB?
        BLE.S   NOBSTARTY4          ; no allora leggi
        SUB.L   #FINETABY-TABY,A0   ; altrimenti torna indietro nella
                                    ; tabella
NOBSTARTY4:
        MOVE.W  (A0),Point4+2       ; copia il valore della coordinata
                                    ; del punto 4 nella variabile apposita
        rts

TABXPOINT:
        dc.l    TABX    ; puntatore alla tabella X

; tabella posizioni X

TABX:
        DC.W    $00D2,$00D2,$00D1,$00D1,$00D0,$00CF,$00CE,$00CD,$00CB,$00C9
        DC.W    $00C8,$00C6,$00C3,$00C1,$00BF,$00BC,$00B9,$00B7,$00B4,$00B1
        DC.W    $00AE,$00AB,$00A8,$00A5,$00A2,$009E,$009B,$0098,$0095,$0092
        DC.W    $008F,$008C,$0089,$0087,$0084,$0081,$007F,$007D,$007A,$0078
        DC.W    $0077,$0075,$0073,$0072,$0071,$0070,$006F,$006F,$006E,$006E
        DC.W    $006E,$006E,$006F,$006F,$0070,$0071,$0072,$0073,$0075,$0077
        DC.W    $0078,$007A,$007D,$007F,$0081,$0084,$0087,$0089,$008C,$008F
        DC.W    $0092,$0095,$0098,$009B,$009E,$00A2,$00A5,$00A8,$00AB,$00AE
        DC.W    $00B1,$00B4,$00B7,$00B9,$00BC,$00BF,$00C1,$00C3,$00C6,$00C8
        DC.W    $00C9,$00CB,$00CD,$00CE,$00CF,$00D0,$00D1,$00D1,$00D2,$00D2

FINETABX:

TABYPOINT:
        dc.l    TABY    ; puntatore alla tabella Y

TABY:
        DC.W    $0081,$0084,$0087,$008A,$008D,$0090,$0093,$0096,$0098,$009B
        DC.W    $009E,$00A0,$00A2,$00A5,$00A7,$00A8,$00AA,$00AC,$00AD,$00AE
        DC.W    $00AF,$00B0,$00B0,$00B1,$00B1,$00B1,$00B1,$00B0,$00B0,$00AF
        DC.W    $00AE,$00AD,$00AC,$00AA,$00A8,$00A7,$00A5,$00A2,$00A0,$009E
        DC.W    $009B,$0098,$0096,$0093,$0090,$008D,$008A,$0087,$0084,$0081
        DC.W    $007D,$007A,$0077,$0074,$0071,$006E,$006B,$0068,$0066,$0063
        DC.W    $0060,$005E,$005C,$0059,$0057,$0056,$0054,$0052,$0051,$0050
        DC.W    $004F,$004E,$004E,$004D,$004D,$004D,$004D,$004E,$004E,$004F
        DC.W    $0050,$0051,$0052,$0054,$0056,$0057,$0059,$005C,$005E,$0060
```

```
        DC.W    $0063,$0066,$0068,$006B,$006E,$0071,$0074,$0077,$007A,$007D
FINETABY:

; Qui sono memorizzate i volta in volta le coordinate dei punti del poligono

Point1: dc.w    100,20
Point2: dc.w    200,20
Point3: dc.w    200,40
Point4: dc.w    100,40


;***************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempmento. Il tipo di riempimento
; e‘ specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo rettangolo da fillare
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;***************************************************************************

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry

fill_esclusivo:
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2

fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #0,$64(a5)              ; BLTAMOD larghezza 20 words (40-40=0)
        move.w  #0,$66(a5)              ; BLTDMOD (40-40=0)

        move.l  a0,$50(a5)              ; BLTAPT - indirizzo al rettangolo
                                        ; il rettangolo sorgente racchiude
                                        ; interamente il poligono
                                        ; puntiamo l’ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$54(a5)              ; BLTDPT - indirizzo rettangolo
        move.w  #(64*100)+20,$58(a5)    ; BLTSIZE (via al blitter !)
```

```
                                        ; larghezza 20 words
        rts                             ; altezza 100 righe (1 plane)


;*****************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

; costanti

DL_Fill         =       1               ; 0=NOFILL / 1=FILL

        IFEQ    DL_Fill
DL_MInterns     =       $CA
        ELSE
DL_MInterns     =       $4A
        ENDC


DrawLine:
        sub.w   d1,d3   ; D3=Y2-Y1

        IFNE    DL_Fill
        beq.s   .end    ; per il fill non servono linee orizzontali
        ENDC

        bgt.s   .y2gy1  ; salta se positivo..
        exg     d0,d2   ; ..altrimenti scambia i punti
        add.w   d3,d1   ; mette in D1 la Y piu‘ piccola
        neg.w   d3      ; D3=DY
.y2gy1:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0
        moveq   #0,d1           ; D1 indice nella tabella ottanti
        sub.w   d0,d2           ; D2=X2-X1
        bge.s   .xdpos          ; salta se positivo..
        addq.w  #2,d1           ; ..altrimenti sposta l'indice
        neg.w   d2              ; e rendi positiva la differenza
.xdpos:
        moveq   #$f,d4          ; maschera per i 4 bit bassi
        and.w   d0,d4           ; selezionali in D4

        IFNE    DL_Fill         ; queste istruzioni vengono assemblate
                                ; solo se DL_Fill=1
        move.b  d4,d5           ; calcola numero del bit da invertire
        not.b   d5              ; (la BCHG numera i bit in modo inverso
        ENDC

        lsr.w   #3,d0           ; offset X:
                                ; Allinea a byte (serve per BCHG)
        add.w   d0,a0           ; aggiunge all'indirizzo
                                ; nota che anche se l'indirizzo
                                ; e‘ dispari non fa nulla perche‘
                                ; il blitter non tiene conto del
                                ; bit meno significativo di BLTxPT
```

```
        ror.w   #4,d4           ; D4 = valore di shift A
        or.w    #$B00+DL_MInterns,d4    ; aggiunge l'opportuno
                                        ; Minterm (OR o EOR)
        swap    d4              ; valore di BLTCON0 nella word alta

        cmp.w   d2,d3           ; confronta DiffX e DiffY
        bge.s   .dygdx          ; salta se >=0..
        addq.w  #1,d1           ; altrimenti setta il bit 0 del'indice
        exg     d2,d3           ; e scambia le Diff
.dygdx:
        add.w   d2,d2           ; D2 = 2*DiffX
        move.w  d2,d0           ; copia in D0
        sub.w   d3,d0           ; D0 = 2*DiffX-DiffY
        addx.w  d1,d1           ; moltiplica per 2 l'indice e
                                ; contemporaneamente aggiunge il flag
                                ; X che vale 1 se 2*DiffX-DiffY<0
                                ; (settato dalla sub.w)
        move.b  Oktants(PC,d1.w),d4     ; legge l'ottante
        swap    d2                      ; valore BLTBMOD in word alta
        move.w  d0,d2                   ; word bassa D2=2*DiffX-DiffY
        sub.w   d3,d2                   ; word bassa D2=2*DiffX-2*DiffY
        moveq   #6,d1                   ; valore di shift e di test per
                                        ; la wait blitter

        lsl.w   d1,d3           ; calcola il valore di BLTSIZE
        add.w   #$42,d3

        lea     $52(a5),a1      ; A1 = indirizzo BLTAPTL
                                ; scrive alcuni registri
                                ; consecutivamente con delle
                                ; MOVE #XX,(Ax)+

        btst    d1,2(a5)        ; aspetta il blitter
.wb:
        btst    d1,2(a5)
        bne.s   .wb

        IFNE    DL_Fill         ; questa istruzione viene assemblata
                                ; solo se DL_Fill=1
        bchg    d5,(a0)         ; Inverte il primo bit della linea
        ENDC

        move.l  d4,$40(a5)      ; BLTCON0/1
        move.l  d2,$62(a5)      ; BLTBMOD e BLTAMOD
        move.l  a0,$48(a5)      ; BLTCPT
        move.w  d0,(a1)+        ; BLTAPTL
        move.l  a0,(a1)+        ; BLTDPT
        move.w  d3,(a1)         ; BLTSIZE
.end:
        rts

;
; se vogliamo eseguire linee per il fill, il codice ottante setta ad 1 il bit
; SING attraverso la costante SML

        IFNE    DL_Fill
SML             =       2
        ELSE
SML             =       0
        ENDC

; tabella ottanti
```

```
Oktants:
        dc.b    SML+1,SML+1+$40
        dc.b    SML+17,SML+17+$40
        dc.b    SML+9,SML+9+$40
        dc.b    SML+21,SML+21+$40

;******************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;******************************************************************************

InitLine:
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;******************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;******************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern linee
        rts

;******************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
;******************************************************************************

CancellaSchermo:
        move.l  #bitplane+78*40,a0      ; indirizzo area da cancellare

        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                 ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0000,$66(a5)          ; BLTDMOD=0
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*100)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo
        rts


;******************************************************************************
```

```
        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$000      ; color0
        dc.w    $0182,$eee      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        Section IlMioPlane,bss_C

BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

;***************************************************************************

In questo esempio realizziamo un poligono che ruota.
Il poligono e‘ formato da 4 punti la cui posizione viene modificata ad ogni
frame leggendola da una tabella precalcolata. Questa tecnica comporta un grande
spreco di memoria. Piu‘ in la‘ nel corso vedremo come calcolare le coordinate
dei punti mediante formule matematiche.
Per disegnare il poligono e‘ sufficente disegnare i lati e fillare. Prima
delle operazioni di disegno e‘ ovviamente necessario cancellare lo schermo
con il blitter.
L’area di schermo da cancellare e quella da fillare sono state calcolate
in modo da comprendere sempre tutto il poligono.
```

26.20 Lezione10x

```
; Lezione10x.s  Effetto morphing!!!!!!! Coded by Executor/RAM JAM

        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup1.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA
```

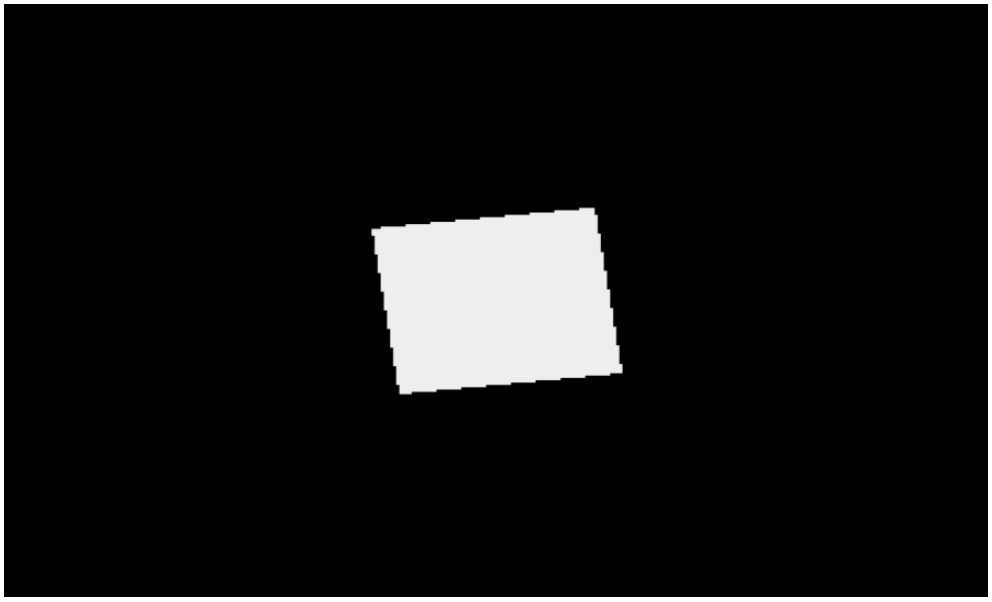


Figura 26.17: Lezione 10v

```
START:
        lea     $dff000,a5              ; CUSTOM REGISTER in a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1

        bsr.w   ScambiaBuffer   ; questa routine scambia i 2 buffer

        bsr.w   CancellaSchermo ; cancella buffer disegno

        bsr.w   DO_IT           ; routine disegno linee
        bsr.w   DO_ANIM         ; routine che gestisce l'animazione

        moveq   #0,d0                   ; esclusivo
        moveq   #0,d1                   ; CARRYIN=0
        move.l  draw_buffer(pc),a0
        bsr.w   FILL                    ; fill buffer disegno

        bsr.w   MORPH           ; calcola il prossimo frame

        btst    #6,$bfe001      ; mouse premuto?
```

```
        bne.s   mouse
        rts


;*****************************************************************************
; Questa routine effettua il disegno della linea. prende come parametri gli
; estremi della linea P1 e P2, e l'indirizzo del bitplane su cui disegnarla.
; D0 - X1 (coord. X di P1)
; D1 - Y1 (coord. Y di P1)
; D2 - X2 (coord. X di P2)
; D3 - Y2 (coord. Y di P2)
; A0 - indirizzo bitplane
;*****************************************************************************

; costanti

DL_Fill         =       1       ; 0=NOFILL / 1=FILL

        IFEQ    DL_Fill
DL_MInterns     =       $CA
        ELSE
DL_MInterns     =       $4A
        ENDC


DrawLine:
        sub.w   d1,d3   ; D3=Y2-Y1

        IFNE    DL_Fill
        beq.s   .end    ; per il fill non servono linee orizzontali
        ENDC

        bgt.s   .y2gy1  ; salta se positivo..
        exg     d0,d2   ; ..altrimenti scambia i punti
        add.w   d3,d1   ; mette in D1 la Y piu' piccola
        neg.w   d3      ; D3=DY
.y2gy1:
        mulu.w  #40,d1          ; offset Y
        add.l   d1,a0
        moveq   #0,d1           ; D1 indice nella tabella ottanti
        sub.w   d0,d2           ; D2=X2-X1
        bge.s   .xdpos          ; salta se positivo..
        addq.w  #2,d1           ; ..altrimenti sposta l'indice
        neg.w   d2              ; e rendi positiva la differenza
.xdpos:
        moveq   #$f,d4          ; maschera per i 4 bit bassi
        and.w   d0,d4           ; selezionali in D4

        IFNE    DL_Fill         ; queste istruzioni vengono assemblate
                                ; solo se DL_Fill=1
        move.b  d4,d5           ; calcola numero del bit da invertire
        not.b   d5              ; (la BCHG numera i bit in modo inverso
        ENDC

        lsr.w   #3,d0           ; offset X:
                                ; Allinea a byte (serve per BCHG)
        add.w   d0,a0           ; aggiunge all'indirizzo
                                ; nota che anche se l'indirizzo
                                ; e' dispari non fa nulla perche'
                                ; il blitter non tiene conto del
                                ; bit meno significativo di BLTxPT
```

```
        ror.w   #4,d4           ; D4 = valore di shift A
        or.w    #$B00+DL_MInterns,d4    ; aggiunge l'opportuno
                                        ; Minterm (OR o EOR)
        swap    d4              ; valore di BLTCON0 nella word alta

        cmp.w   d2,d3           ; confronta DiffX e DiffY
        bge.s   .dygdx          ; salta se >=0..
        addq.w  #1,d1           ; altrimenti setta il bit 0 del'indice
        exg     d2,d3           ; e scambia le Diff
.dygdx:
        add.w   d2,d2           ; D2 = 2*DiffX
        move.w  d2,d0           ; copia in D0
        sub.w   d3,d0           ; D0 = 2*DiffX-DiffY
        addx.w  d1,d1           ; moltiplica per 2 l'indice e
                                ; contemporaneamente aggiunge il flag
                                ; X che vale 1 se 2*DiffX-DiffY<0
                                ; (settato dalla sub.w)
        move.b  Oktants(PC,d1.w),d4     ; legge l'ottante
        swap    d2                      ; valore BLTBMOD in word alta
        move.w  d0,d2                   ; word bassa D2=2*DiffX-DiffY
        sub.w   d3,d2                   ; word bassa D2=2*DiffX-2*DiffY
        moveq   #6,d1                   ; valore di shift e di test per
                                        ; la wait blitter

        lsl.w   d1,d3           ; calcola il valore di BLTSIZE
        add.w   #$42,d3

        lea     $52(a5),a1      ; A1 = indirizzo BLTAPTL
                                ; scrive alcuni registri
                                ; consecutivamente con delle
                                ; MOVE #XX,(Ax)+

        btst    d1,2(a5)        ; aspetta il blitter
.wb:
        btst    d1,2(a5)
        bne.s   .wb

        IFNE    DL_Fill         ; questa istruzione viene assemblata
                                ; solo se DL_Fill=1
        bchg    d5,(a0)         ; Inverte il primo bit della linea
        ENDC

        move.l  d4,$40(a5)      ; BLTCON0/1
        move.l  d2,$62(a5)      ; BLTBMOD e BLTAMOD
        move.l  a0,$48(a5)      ; BLTCPT
        move.w  d0,(a1)+        ; BLTAPTL
        move.l  a0,(a1)+        ; BLTDPT
        move.w  d3,(a1)         ; BLTSIZE
.end:
        rts

;
; se vogliamo eseguire linee per il fill, il codice ottante setta ad 1 il bit
; SING attraverso la costante SML

        IFNE    DL_Fill
SML             =       2
        ELSE
SML             =       0
        ENDC

; tabella ottanti
```

```
Oktants:
        dc.b    SML+1,SML+1+$40
        dc.b    SML+17,SML+17+$40
        dc.b    SML+9,SML+9+$40
        dc.b    SML+21,SML+21+$40

;******************************************************************************
; Questa routine setta i registri del blitter che non devono essere
; cambiati tra una line e l'altra
;******************************************************************************

InitLine:
        btst    #6,2(a5) ; dmaconr
WBlit_Init:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Init

        moveq   #-1,d5
        move.l  d5,$44(a5)              ; BLTAFWM/BLTALWM = $FFFF
        move.w  #$8000,$74(a5)          ; BLTADAT = $8000
        move.w  #40,$60(a5)             ; BLTCMOD = 40
        move.w  #40,$66(a5)             ; BLTDMOD = 40
        rts

;******************************************************************************
; Questa routine definisce il pattern che deve essere usato per disegnare
; le linee. In pratica si limita a settare il registro BLTBDAT.
; D0 - contiene il pattern della linea
;******************************************************************************

SetPattern:
        btst    #6,2(a5) ; dmaconr
WBlit_Set:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   Wblit_Set

        move.w  d0,$72(a5)      ; BLTBDAT = pattern linee
        rts


;******************************************************************************
; Questa routine scambia i 2 buffer scambiando gli indirizzi nelle
; variabili VIEW_BUFFER e  DRAW_BUFFER.
; Inoltre aggiorna nella copperlist le istruzioni che caricano i registri
; BPLxPT, in modo che puntino al nuovo buffer da visualizzare.
;******************************************************************************

ScambiaBuffer:
        move.l  draw_buffer(pc),d0              ; scambia il contenuto
        move.l  view_buffer(pc),draw_buffer     ; delle variabili
        move.l  d0,view_buffer                  ; in d0 c'e' l'indirizzo
                                                ; del nuovo buffer
                                                ; da visualizzare

; aggiorna la copperlist puntando i bitplanes del nuovo buffer da visualizzare

        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        rts
```

```
; puntatori ai 2 buffer

view_buffer:    dc.l    BITPLANE1       ; buffer visualizzato
draw_buffer:    dc.l    BITPLANE1b      ; buffer di disegno

;***************************************************************************
; Questa routine cancella lo schermo mediante il blitter.
; l'indirizzo dello schermo viene letto dalla variabile DRAW_BUFFER
;***************************************************************************

CancellaSchermo:
        move.l  draw_buffer(pc),a0      ; indirizzo buffer disegno

        btst    #6,2(a5)
WBlit3:
        btst    #6,2(a5)                ; attendi che il blitter abbia finito
        bne.s   wblit3

        move.l  #$01000000,$40(a5)      ; BLTCON0 e BLTCON1: Cancella
        move.w  #$0000,$66(a5)          ; BLTDMOD=0
        move.l  a0,$54(a5)              ; BLTDPT
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; cancella tutto lo schermo
        rts

;***************************************************************************
; Questa routine copia un rettangolo di schermo da una posizione fissa
; ad un indirizzo specificato come parametro. Il rettangolo di schermo che
; viene copiato racchiude interamente le 2 linee.
; Durante la copia viene effettuato anche il riempimento. Il tipo di
; riempimento e' specificato tramite i parametri.
; I parametri sono:
; A0 - indirizzo rettangolo da fillare
; D0 - se vale 0 allora effettua fill inclusivo, altrimenti fa fill esclusivo
; D1 - se vale 0 allora effettua FILL_CARRYIN=0, altrimenti FILL_CARRYIN=1
;***************************************************************************

Fill:
        btst    #6,2(a5) ; dmaconr
WBlit1:
        btst    #6,2(a5) ; dmaconr - attendi che il blitter abbia finito
        bne.s   wblit1

        move.w  #$09f0,$40(a5)          ; BLTCON0 copia normale

        tst.w   d0                      ; testa D0 per decidere il tipo di fill
        bne.s   fill_esclusivo
        move.w  #$000a,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill inclusivo e del modo discendente
        bra.s   test_fill_carry

fill_esclusivo:
        move.w  #$0012,d2               ; valore di BLTCON1: settati i bit del
                                        ; fill esclusivo e del modo discendente

test_fill_carry:
        tst.w   d1                      ; testa D1 per vedere se deve settare
                                        ; il bit FILL_CARRYIN

        beq.s   fatto_bltcon1           ; se D1=0 salta..
        bset    #2,d2                   ; altrimenti setta il bit 2 di D2
```

```
fatto_bltcon1:
        move.w  d2,$42(a5)              ; BLTCON1

        move.w  #0,$64(a5)              ; BLTAMOD larghezza 20 words (40-40=0)
        move.w  #0,$66(a5)              ; BLTDMOD (40-40=0)

        lea     40*256-2(a0),a0         ; puntiamo l'ultima word del rettangolo
                                        ; per via del modo discendente

        move.l  a0,$50(a5)              ; BLTAPT - indirizzo al rettangolo
                                        ; il rettangolo sorgente racchiude
                                        ; interamente il poligono

        move.l  a0,$54(a5)              ; BLTDPT - indirizzo rettangolo
        move.w  #(64*256)+20,$58(a5)    ; BLTSIZE (via al blitter !)
                                        ; larghezza 20 words
                                        ; altezza 256 righe (1 plane)
        rts

*****************************************************************************
; Questa routine calcola i punti che costituiscono un frame intermedio.
; Il morph si realizza nel modo seguente: per ogni punto si prendono le
; coordinate relative nel frame di partenza (sorgente) e in quello di arrivo
; (destinazione) e si calcolano la differenze. Tali differenze sono la distanza
; (rispettivamente lungo le X e lungo le Y) che il punto deve percorrere per
; andare dalla posizione nel frame di partenza a quella nel frame di arrivo.
; Questa distanza dovra' essere percorsa in un numero variabile di frames che
; viene memorizzato nella variabile MAXSTEP. Dividendo la distanza per il
; numero di frames otteniamo la quantita' di spazio che il punto deve
; percorrere ad ogni frame. Moltiplicando tale quantita' per il numero di
; frame disegnati, e aggiungendoci la posizione di partenza otteniamo la
; posizione attuale del punto.
; La formula per calcolare il valore della coordinata X nel frame attuale
; e' dunque la seguente:

; X_attuale = X_partenza+(X_arrivo-X_partenza)*Frame_attuale/Numero_frames.

; La formula per la Y e' la stessa. Le coordinate X_arrivo e X_partenza sono
; memorizzate nei dati di ogni frame, mentre Frame_attuale e Numero_frames sono
; memorizzati rispettivamente nelle variabili STEP e MAXSTEP. Queste formule
; vengono applicate dalla routine "MORPH" alle coordinate di tutti i punti che
; costituiscono un frame.
*****************************************************************************

MORPH:
        moveq   #24-1,d7        ; 24 punti per frame

        move.l  MAXSTEP(PC),d5  ; d5 = numero totale di frame intermedi

        move.l  STEP(PC),d6     ; d6 = contatore frame attuale

        lea     POINTS(PC),a2   ; A2 = vettore punti
        move.l  SOURFRM(PC),a0
        move.l  (a0),a0         ; A0 = frame di partenza
        move.l  DESTFRM(PC),a1
        move.l  (a1),a1         ; A1 = frame di arrivo

; Calcola le coordinate dei punti del frame intermedio

ProcX:
        moveq   #0,d0
```

```
        move.w  (a1)+,d0        ; d0 = x1
        sub.w   (a0),d0         ; d0 = x1-x0
        muls.w  d6,d0           ; d0 = (x1-x0)·d6
        divs.w  d5,d0           ; d0 = (x1-x0)·d6/d5
        add.w   (a0)+,d0        ; d0 = x0+(x1-x0)·d6/d5
ProcY:
        moveq   #0,d1
        move.w  (a1)+,d1        ; d1 = y1
        sub.w   (a0),d1         ; d1 = y1-y0
        muls.w  d6,d1           ; d1 = (y1-y0)·d6
        divs.w  d5,d1           ; d1 = (y1-y0)·d6/d5
        add.w   (a0)+,d1        ; d1 = y0+(y1-y0)·d6/d5

        move.w  d0,(a2)+        ; memorizza coordinate frame intermedio
        move.w  d1,(a2)+

        dbra    d7,ProcX        ; Esegui per tutti i punti
        addq.l  #1,STEP         ; aumenta contatore frame attuale
        rts


*****************************************************************************
; Routine che gestisce l'animazione
*****************************************************************************

DO_ANIM:
        tst.l   FX              ; testa contatore
        bne.s   DO_AM           ; se non vale 0 salta..

        move.l  SCRIPT(PC),a0   ; ..altrimenti leggi indirizzo del punto
                                ; dell'animazione in cui ci troviamo.
AT:
        tst.l   (a0)
        bne.s   DO_X
        tst.l   4(a0)
        bne.s   DO_X            ; se i puntatori ai frames sono diversi da 0
                                ; salta..

RESTART:
        lea     STORY(PC),a0    ; altrimenti ricomincia l'animazione da capo
        bra.s   AT

DO_X:
        move.l  (a0)+,SOURFRM   ; legge nuovo frame sorgente
        move.l  (a0)+,DESTFRM   ; legge nuovo frame destinazione
        move.l  (a0)+,MAXSTEP   ; nuovo numero di frames intermedi
        move.l  MAXSTEP(PC),FX  ; copialo nel contatore
        move.l  a0,SCRIPT       ; memorizza punto di arrivo animazione
        move.l  #1,STEP         ; inizializza contatore frame intermedio
DO_AM:
        subq.l  #1,FX           ; decrementa contatore
        rts


*****************************************************************************
; Questa routine disegna le linee che formano il frame attuale
*****************************************************************************

DO_IT:
        lea     POINTS(PC),a3   ; vettore punti frame attuale
        moveq   #24-1,d7        ; 24 punti per ogni frame
        bsr.w   InitLine        ; inizializza line-mode
```

```
        move.w  #$ffff,d0       ; linea continua
        bsr.w   SetPattern      ; definisce pattern

        move.l  (a3),-(a7)      ; memorizza coordinate al primo punto
AnimLoop:
        movem.w (a3)+,d0-d1     ; legge coordinate i-esimo punto
        movem.w (a3),d2-d3      ; legge coordinate (i+1)-esimo punto
        tst.w   d7
        bne.s   NoLast          ; se non siamo all'ultimo punto salta..
        movem.w (a7)+,d2-d3     ; ..altrimenti, ultimo punto = primo punto
NoLast:
        move.l  draw_buffer(pc),a0      ; ind. buffer disegno
        bsr.w   DrawLine                ; disegna linea
        dbra    d7,AnimLoop
        rts

*************************
; Variabili
*************************

; Puntatore alla fase attuale dell'animazione

SCRIPT: dc.l    STORY

; contatore per segnalare il passagio da una fase all'altra

FX:     dc.l    0

; numero di frame intermedi tra la sorgente e la destinazione

MAXSTEP:        dc.l    200

; numero frame intermedio attuale

STEP:           dc.l    1

; indirizzo frame sorgente

SOURFRM:        dc.l    FRM1

; indirizzo frame destinazione

DESTFRM:        dc.l    FRM2

; Animazione: ogni fase ha un frame sorgente, un frame destinazione, e
; il tempo di morphing, ovvero il numero di frame intermedi che devono
; essere generati. Per es. la prima fase va dal frame FRM1 al frame FRM2
; in 24 frames intermedi.

STORY:
        dc.l    FRM1,FRM2,24
        dc.l    FRM2,FRM3,24
        dc.l    FRM3,FRM4,24
        dc.l    FRM4,FRM5,24
        dc.l    FRM5,FRM6,24
        dc.l    FRM6,FRM7,24
        dc.l    FRM7,FRM8,24
        dc.l    FRM8,FRM9,24
        dc.l    FRM9,FRM10,24
        dc.l    FRM10,FRM11,24
        dc.l    FRM11,FRM12,24
```

```
        dc.l    FRM12,FRM13,24
        dc.l    FRM13,FRM14,24
        dc.l    FRM14,FRM15,24
        dc.l    FRM15,FRM16,12
        dc.l    FRM16,FRM17,12
        dc.l    FRM17,FRM18,24
        dc.l    FRM18,FRM19,24
        dc.l    FRM19,FRM20,24
        dc.l    FRM20,FRM21,24
        dc.l    FRM21,FRM22,24
        dc.l    FRM22,FRM23,24
        dc.l    FRM23,FRM23,48
        dc.l    FRM23,FRM24,24
        dc.l    FRM24,FRM25,24
        dc.l    FRM25,FRM26,24
        dc.l    FRM26,FRM27,24
        dc.l    FRM27,FRM26,24
        dc.l    FRM26,FRM27,24
        dc.l    FRM27,FRM26,24
        dc.l    FRM26,FRM27,24
        dc.l    FRM27,FRM28,24
        dc.l    FRM28,FRM29,24
        dc.l    FRM29,FRM30,96
        dc.l    FRM30,FRM31,24
        dc.l    FRM31,FRM32,24
        dc.l    FRM32,FRM31,24

        dc.l    0,0                     ; fine

; Vettore che contiene le coordinate del frame intermedio

POINTS:
        dcb.b   24*4,0

*****************************************************************************
; Dati dei frames: per ogni frame ci sono le coordinate X e Y dei punti
; che lo formano.
*****************************************************************************

FRAMES:
FRAME1:
        dc.w    $13F,$7E,$13F,$AE,$13F,$FF,$11C,$FF,$DA,$FF,$A5
        dc.w    $FF,$87,$FF,$6E,$FF,$4A,$FF,$2E,$FF,$13,$FF,0,$FF
        dc.w    0,$C8,0,$B6,0,$A5,0,$82,0,$38,0,0,$15,0,$7E,0,$95
        dc.w    0,$DF,0,$13F,0,$13F,$37
FRAME2:
        dc.w    $C9,$68,$C7,$76,$C2,$82,$BA,$8D,$AF,$95,$A3,$9A
        dc.w    $96,$9B,$88,$9A,$7C,$95,$71,$8D,$69,$82,$64,$76
        dc.w    $63,$68,$64,$5B,$69,$4F,$71,$44,$7C,$3C,$88,$37
        dc.w    $96,$35,$A3,$37,$AF,$3C,$BA,$44,$C2,$4F,$C7,$5B
FRAME3:
        dc.w    $13F,$6F,$C7,$76,$C2,$82,$BA,$8D,$AF,$95,$A3,$9A
        dc.w    $95,$FF,$88,$9A,$7C,$95,$71,$8D,$69,$82,$64,$76,0
        dc.w    $69,$64,$5B,$69,$4F,$71,$44,$7C,$3C,$88,$37,$94,0
        dc.w    $A3,$37,$AF,$3C,$BA,$44,$C2,$4F,$C7,$5B
FRAME4:
        dc.w    $13F,$6A,$6E,$4F,$69,$5B,$61,$66,$56,$6E,$4A,$73
        dc.w    $9D,$FF,$2F,$73,$23,$6E,$18,$66,$10,$5B,11,$4F,0
        dc.w    $82,11,$34,$10,$28,$18,$1D,$23,$15,$2F,$10,$9A,0
        dc.w    $4A,$10,$56,$15,$61,$1D,$69,$28,$6E,$34
FRAME5:
        dc.w    $13F,$5A,$123,$BF,$11E,$CB,$116,$D6,$10B,$DE,$FF
```

```
        dc.w    $E3,$86,$FF,$E4,$E3,$D8,$DE,$CD,$D6,$C5,$CB,$C0
        dc.w    $BF,0,$77,$C0,$A4,$C5,$98,$CD,$8D,$D8,$85,$E4,$80
        dc.w    $84,0,$FF,$80,$10B,$85,$116,$8D,$11E,$98,$123,$A4
FRAME6:
        dc.w    $13F,$8F,$128,$5A,$123,$66,$11B,$71,$110,$79,$104
        dc.w    $7E,$86,$FF,$E9,$7E,$DD,$79,$D2,$71,$CA,$66,$C5
        dc.w    $5A,0,$75,$C5,$3F,$CA,$33,$D2,$28,$DD,$20,$E9,$1B
        dc.w    $91,0,$104,$1B,$110,$20,$11B,$28,$123,$33,$128
        dc.w    $3F
FRAME7:
        dc.w    $13F,$7A,$75,$BE,$70,$CA,$68,$D5,$5D,$DD,$51,$E2
        dc.w    $A7,$FF,$36,$E2,$2A,$DD,$1F,$D5,$17,$CA,$12,$BE,0
        dc.w    $72,$12,$A3,$17,$97,$1F,$8C,$2A,$84,$36,$7F,$A1,0
        dc.w    $51,$7F,$5D,$84,$68,$8C,$70,$97,$75,$A3
FRAME8:
        dc.w    $9D,$77,$C7,$7F,$C2,$8B,$9E,$7B,$AF,$9E,$A3,$A3
        dc.w    $9A,$7C,$88,$A3,$7C,$9E,$97,$7B,$69,$8B,$64,$7F
        dc.w    $96,$78,$64,$64,$69,$58,$96,$74,$7B,$46,$88,$40
        dc.w    $9A,$73,$A3,$40,$AF,$45,$9B,$75,$C2,$58,$C7,$64
FRAME9:
        dc.w    $B7,$87,$111,$98,$106,$B2,$B9,$8F,$DD,$DA,$C4,$E5
        dc.w    $B0,$91,$8A,$E5,$70,$DA,$AA,$8F,$47,$B2,$3D,$98
        dc.w    $A8,$88,$3D,$5E,$47,$44,$A8,$81,$6E,$1D,$8A,$11
        dc.w    $B0,$7E,$C4,$11,$DD,$1B,$B3,$83,$106,$44,$111,$5E
FRAME10:
        dc.w    $AE,$86,$C4,$89,$C1,$8F,$AF,$87,$B7,$98,$B1,$9B
        dc.w    $AD,$87,$A5,$9B,$9F,$98,$AC,$87,$95,$8F,$93,$89
        dc.w    $AC,$86,$93,$7C,$95,$76,$AC,$84,$9E,$6D,$A5,$6A
        dc.w    $AD,$84,$B1,$6A,$B7,$6C,$AE,$85,$C1,$76,$C4,$7C
FRAME11:
        dc.w    $F4,$7F,$EB,$8E,$E0,$A3,$E1,$C2,$BF,$C1,$AB,$CB
        dc.w    $9B,$D2,$87,$CB,$72,$C1,$52,$C9,$52,$A3,$4A,$8E
        dc.w    $49,$85,$4A,$68,$52,$55,$56,$3D,$6F,$37,$87,$2E
        dc.w    $9C,$27,$AB,$2E,$BF,$34,$E3,$3D,$E0,$55,$EB,$68
FRAME12:
        dc.w    $118,$EE,$E6,$F0,$D4,$BB,$91,$7E,$C4,$3C,$B9,$2E
        dc.w    $7B,$2D,$8F,$FE,$6C,$E8,$6B,$A9,$66,$74,$5F,$43
        dc.w    $56,$1F,$3A,$60,$39,$45,$48,12,$79,9,$A2,9,$D6
        dc.w    $11,$EB,$25,$EB,$38,$E9,$63,$A7,$7F,$EB,$AB
FRAME13:
        dc.w    $BA,$A5,$80,$9A,$81,$77,$C3,$7A,$CB,$38,$C0,$2A
        dc.w    $82,$29,$70,$B5,$69,$F9,$52,$F7,$35,$F6,$37,$C6
        dc.w    $36,$82,$41,$5C,$40,$41,$4F,8,$80,5,$A9,5,$DD,13
        dc.w    $F2,$21,$F2,$34,$EE,$5C,$E8,$F6,$C6,$FB
FRAME14:
        dc.w    $D2,$A6,$D5,$85,$D7,$26,$B1,$1D,$A1,$49,$96,$1E
        dc.w    $7B,$24,$66,$B2,$79,$F8,$62,$F6,$52,$EB,$49,$C7
        dc.w    $4A,$A5,$50,$77,$59,$40,$5F,7,$90,4,$B9,4,$ED,12
        dc.w    $102,$20,$102,$33,$FE,$5B,$F8,$F5,$D6,$FA
FRAME15:
        dc.w    $E0,$BA,$E3,$99,$E5,$3A,$CC,$33,$B6,$2F,$85,$31
        dc.w    $62,$39,$46,$5B,$36,$5C,$32,$5D,$38,$1D,$70,$15
        dc.w    $99,$12,$C0,$10,$E6,15,$107,$12,$11D,$18,$11C,$73
        dc.w    $10C,$E1,$8E,$DE,$55,$CB,$5A,$87,$72,$61,$8D,$BF
FRAME16:
        dc.w    $D9,$94,$5F,$90,$60,$78,$D4,$72,$D7,$53,$B5,$35
        dc.w    $85,$35,$58,$4F,$57,$DE,$35,$D3,$43,$35,$54,$24
        dc.w    $7C,$15,$B7,15,$CF,$1C,$ED,$3E,$F5,$61,$F3,$86
        dc.w    $F3,$B7,$F5,$DD,$103,$EC,$D7,$EB,$D9,$CE,$D8,$BB
FRAME17:
        dc.w    $CF,$F4,$C4,$F4,$D0,$77,$D4,$40,$A6,$2B,$93,$56
        dc.w    $7C,$2E,$62,$48,$5E,$F8,$35,$CE,$4D,$2E,$56,$1D
```

```
        dc.w    $66,15,$76,10,$8B,8,$A0,10,$B4,14,$DE,$1D,$F4,$47
        dc.w    $F7,$63,$F5,$AC,$F3,$C9,$EB,$DB,$DE,$EC
FRAME18:
        dc.w    $CA,$F7,$C6,$F0,$BC,$F2,$A9,$F0,$95,$F4,$8A,$F3
        dc.w    $7D,$F4,$72,$F3,$68,$F7,$3F,$CD,$57,$2D,$60,$1C
        dc.w    $70,14,$80,9,$95,7,$AA,9,$BE,13,$DD,$2C,$E6,$5B
        dc.w    $E7,$7F,$DD,$BA,$DC,$D3,$DA,$E3,$D0,$EE
FRAME19:
        dc.w    $C8,$EB,$BF,$DD,$B6,$C4,$A9,$F0,$91,$CC,$8A,$F3
        dc.w    $7D,$F4,$5C,$86,0,$8B,0,$62,$5F,$6E,$5F,$19,$70
        dc.w    14,$88,$21,$95,7,$A7,$25,$BE,13,$E8,$67,$13F,$5E
        dc.w    $13F,$8C,$E6,$7E,$DC,$D3,$DA,$E3,$D0,$EE
FRAME20:
        dc.w    $A1,$A6,$AE,$FF,$78,$FF,$8C,$A7,$8A,$98,$7F,$8F
        dc.w    $6E,$8A,$5C,$86,0,$90,0,$64,$5F,$6E,$79,$7D,$88
        dc.w    $6B,$82,0,$AB,0,$9C,$70,$B2,$7B,$E8,$67,$13F,$5F
        dc.w    $13F,$8D,$E6,$7E,$AC,$97,$A8,$98,$A0,$99
FRAME21:
        dc.w    $54,$D5,$AE,$FF,$78,$FF,$53,$EA,$42,$D4,$2B,$D1
        dc.w    $3A,$BB,$3A,$B4,0,$90,0,$64,$45,$AC,$52,$B1,$56
        dc.w    $AB,$82,0,$AB,0,$61,$A8,$56,$BB,$62,$BA,$13F,$5F
        dc.w    $13F,$8D,$D4,$AE,$86,$B9,$69,$C0,$4E,$CE
FRAME22:
        dc.w    $B8,$66,$AE,$FF,$78,$FF,$AB,$60,$AD,$4F,$B0,$47
        dc.w    $9D,$52,$97,$57,0,$90,0,$64,$91,$4D,$A0,$43,$A7
        dc.w    $2B,$82,0,$AB,0,$B3,$27,$BE,$36,$DB,$57,$13F,$5F
        dc.w    $13F,$8D,$CF,$63,$C8,$5C,$C2,$54,$BC,$45
FRAME23:
        dc.w    $A6,$96,$A6,$B5,$8A,$B5,$7D,$B5,$63,$B5,$3E,$B5
        dc.w    $25,$B5,0,$B5,0,$90,0,$64,0,$5D,0,$3D,0,$2F,0,0
        dc.w    $A6,0,$A6,$27,$A6,$36,$A6,$48,$A6,$52,$A6,$5C,$A6
        dc.w    $62,$A6,$68,$A6,$78,$A6,$84
FRAME24:
        dc.w    $AF,$A8,$BD,$BB,$E6,$7E,$EF,$86,$E0,$BB,$C2,$F5
        dc.w    $A4,$F9,$59,$F8,$41,$A1,$24,$4F,$2E,$42,$3B,$49
        dc.w    $57,$88,$5B,$10,$6F,$10,$79,$77,$8B,3,$96,3,$9F
        dc.w    $11,$9A,$78,$BF,$18,$D0,$1E,$CE,$31,$AD,$8D
FRAME25:
        dc.w    $B4,$AB,$BD,$BB,$BB,$93,$CF,$90,$D4,$C1,$C2,$F5
        dc.w    $A4,$F9,$59,$F8,$41,$A1,$39,$81,$42,$78,$50,$77
        dc.w    $57,$88,$5A,$68,$75,$62,$79,$77,$8B,3,$96,3,$9F
        dc.w    $11,$9A,$78,$A2,$69,$B2,$69,$B5,$6D,$B4,$8D
FRAME26:
        dc.w    $D6,$4A,$D1,$4B,$C8,$4C,$B9,$52,$A1,$5F,$9C,$63
        dc.w    $90,$5C,$82,$56,$79,$51,$6B,$4D,$5D,$4C,$56,$44
        dc.w    $5E,$43,$71,$44,$80,$4A,$8B,$51,$94,$5A,$9D,$53
        dc.w    $A6,$59,$B4,$4D,$C0,$46,$CF,$42,$D7,$42,$E2,$44
FRAME27:
        dc.w    $DB,$68,$CD,$6C,$C1,$6C,$B5,$68,$A3,$5D,$9B,$5D
        dc.w    $95,$65,$8B,$6A,$81,$6B,$7B,$6B,$68,$66,$63,$5D
        dc.w    $75,$62,$81,$63,$88,$62,$93,$5B,$96,$58,$9F,$51
        dc.w    $A8,$57,$B2,$5D,$BC,$63,$C8,$64,$D6,$62,$E2,$60
FRAME28:
        dc.w    $BB,$7B,$BA,$8C,$AE,$A3,$9F,$B8,$77,$E3,$4C,$FC
        dc.w    $7E,$D1,$98,$B4,$AB,$9A,$B0,$8D,$B1,$7B,$AC,$64
        dc.w    $9F,$5C,$93,$48,$92,$35,$92,$20,$9C,13,$AC,8,$BE
        dc.w    13,$C8,$1D,$CE,$36,$CE,$48,$C5,$5C,$BC,$65
FRAME29:
        dc.w    $BA,$57,$C6,$64,$D0,$7D,$D4,$A9,$D9,$C9,$108,$D7
        dc.w    $D7,$D1,$D2,$CC,$C6,$82,$BF,$6B,$B2,$5F,$9D,$52
        dc.w    $8D,$57,$76,$54,$65,$46,$55,$37,$4D,$21,$55,$11
        dc.w    $65,6,$7B,9,$91,$17,$A0,$24,$A7,$37,$A9,$46
```

```
FRAME30:
        dc.w    $C1,$66,$D3,$58,$10B,$A5,$106,$6A,$BD,$3A,$10E
        dc.w    $17,$C7,$3D,$10B,$64,$114,$BA,$D3,$63,$C4,$6F,$97
        dc.w    $85,$94,$92,$83,$A3,$74,$A8,$5F,$AC,$4B,$A9,$43
        dc.w    $9B,$43,$8C,$51,$7E,$63,$72,$74,$6E,$8A,$6F,$93
        dc.w    $76
FRAME31:
        dc.w    $D4,$39,$EF,$61,$106,$5A,$11A,$56,$110,$62,$EB
        dc.w    $6F,$E6,$BF,$DD,$B6,$DA,$70,$BF,$6E,$8E,$6D,$74
        dc.w    $6C,$64,$BB,$55,$B2,$67,$6C,$52,$70,$3E,$6D,$60
        dc.w    $59,$36,$50,$44,$42,$56,$36,$67,$32,$7D,$33,$A1
        dc.w    $32
FRAME32:
        dc.w    $D2,$98,$F0,$C2,$F7,$BA,$108,$B2,$109,$BB,$EC,$D0
        dc.w    $E0,$FF,$D7,$FE,$D0,$CA,$BD,$D6,$8D,$D5,$75,$CD
        dc.w    $6F,$FE,$64,$FF,$68,$CD,$53,$D1,$3C,$BE,$61,$BA
        dc.w    $3D,$BA,$45,$A3,$57,$97,$68,$93,$7B,$9A,$9F,$A4

; indirizzi dei frames

FRM1:   dc.l    FRAME1          ; FRAMES+24*4*0
FRM2:   dc.l    FRAME2          ; FRAMES+24*4*1
FRM3:   dc.l    FRAME3          ; FRAMES+24*4*2
FRM4:   dc.l    FRAME4          ; FRAMES+24*4*3
FRM5:   dc.l    FRAME5          ; FRAMES+24*4*4
FRM6:   dc.l    FRAME6          ; FRAMES+24*4*5
FRM7:   dc.l    FRAME7          ; FRAMES+24*4*6
FRM8:   dc.l    FRAME8          ; FRAMES+24*4*7
FRM9:   dc.l    FRAME9          ; FRAMES+24*4*8
FRM10:  dc.l    FRAME10         ; FRAMES+24*4*9
FRM11:  dc.l    FRAME11         ; FRAMES+24*4*10
FRM12:  dc.l    FRAME12         ; FRAMES+24*4*11
FRM13:  dc.l    FRAME13         ; FRAMES+24*4*12
FRM14:  dc.l    FRAME14         ; FRAMES+24*4*13
FRM15:  dc.l    FRAME15         ; FRAMES+24*4*14
FRM16:  dc.l    FRAME16         ; FRAMES+24*4*15
FRM17:  dc.l    FRAME17         ; FRAMES+24*4*16
FRM18:  dc.l    FRAME18         ; FRAMES+24*4*17
FRM19:  dc.l    FRAME19         ; FRAMES+24*4*18
FRM20:  dc.l    FRAME20         ; FRAMES+24*4*19
FRM21:  dc.l    FRAME21         ; FRAMES+24*4*20
FRM22:  dc.l    FRAME22         ; FRAMES+24*4*21
FRM23:  dc.l    FRAME23         ; FRAMES+24*4*22
FRM24:  dc.l    FRAME24         ; FRAMES+24*4*23
FRM25:  dc.l    FRAME25         ; FRAMES+24*4*24
FRM26:  dc.l    FRAME26         ; FRAMES+24*4*25
FRM27:  dc.l    FRAME27         ; FRAMES+24*4*26
FRM28:  dc.l    FRAME28         ; FRAMES+24*4*27
FRM29:  dc.l    FRAME29         ; FRAMES+24*4*28
FRM30:  dc.l    FRAME30         ; FRAMES+24*4*29
FRM31:  dc.l    FRAME31         ; FRAMES+24*4*30
FRM32:  dc.l    FRAME32         ; FRAMES+24*4*31

;***************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
```

```
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$1200      ; Bplcon0 - 1 bitplane lowres

BPLPOINTERS:
        dc.w    $e0,$0000,$e2,$0000     ;primo   bitplane

        dc.w    $0180,$44b      ; color0
        dc.w    $0182,$f88      ; color1
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

        SECTION bitplane,BSS_C
; buffer 1

BITPLANE1:
        ds.b    40*256

; buffer 2

BITPLANE1b:
        ds.b    40*256

        end

;***************************************************************************
```

In questo esempio presentiamo una routine che esegue un animazione mediante un effetto di morph. Questa e‘ una delle tecniche principali impiegate dalla mitica "State of the art" degli spaceballs, forse la piu‘ famosa demo mai realizzata.
Vediamo come si realizza questa tecnica. Ogni frame dell’animazione e‘ realizzato mediante una singolo poligono fillato. Per disegnare il poligono dobbiamo tracciarne i bordi e poi eseguire un fill con il blitter. Il primo di questi compiti e‘ svolto dalla routine "DO_IT" e il secondo dalla routine "FILL". Per realizzare l’animazione basta cambiare la forma del poligono ad ogni frame, e cio‘ viene fatto cambiando le coordinate X e Y di tutti i suoi vertici. Se memorizzassimo le coordinate di tutti i punti per tutti i frames, occuperemmo una quantita‘ ENORME di memoria. Utiliziamo allora un metodo piu‘ furbo: memorizziamo solo alcuni frames, e calcoliamo di volta in volta gli altri facendo il morph tra 2 dei frame che abbiamo memorizzato.
L’animazione, dunque, e‘ divisa in varie fasi, durante le quali viene effettuato il morph tra un frame sorgente e un frame destinazione in un certo numero di frames. Quando viene raggiunto il frame destinazione, si passa ad una nuova fase, cambiando frame sorgente, frame destinazione e numero di frame impiegati dal morph. Le fasi dell’animazione sono memorizzate in sequenza a partire dall’indirizzo "STORY", e il passaggio da una fase ell’altra viene gestito dalla routine "DO_ANIM"

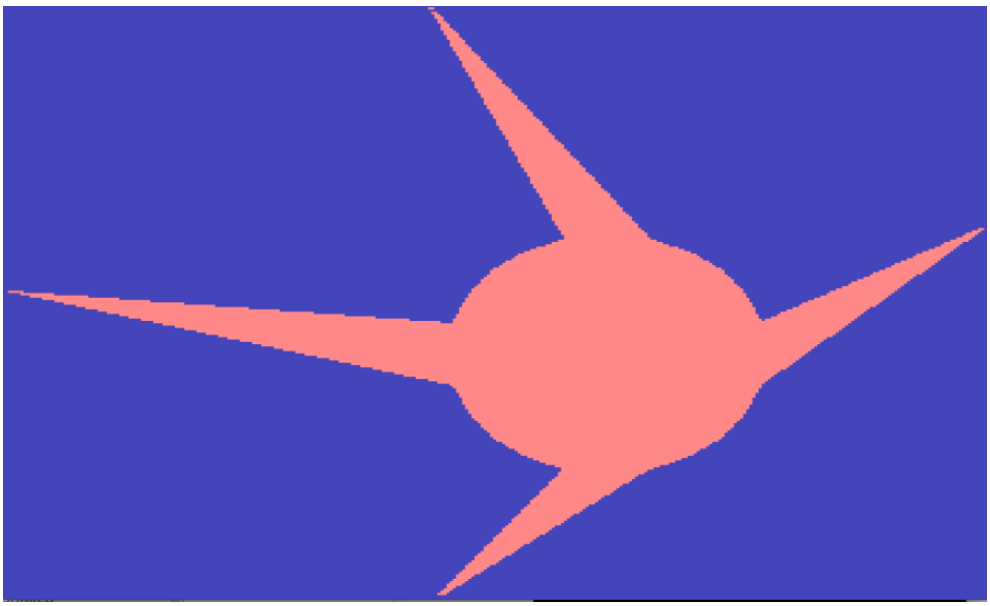


Figura 26.18: Lezione 10x

CAPITOLO 27

LEZIONE 11

27.1 Lezione11a

```
; Lezione11a.s          Esecuzione di un paio di istruzioni privilegiate.

Inizio:
        move.l  4.w,a6                  ; ExecBase in a6
        lea     SuperCode(PC),a5        ; Routine da eseguire in supervisor
        jsr     -$1e(a6)                ; LvoSupervisor - esegui la routine
                                        ; (non salva i registri! attenzione!)
        rts                             ; esci, dopo aver eseguito la routine
                                        ; "SuperCode" in supervisor.

; Routine eseguita in modo supervisore
;         --
;         \/
;       -    -
;
;        /  \

SuperCode:
        move.w  SR,d0           ; istruzione privilegiata
        move.w  d0,sr           ; istruzione privilegiata
        RTE     ; Return From Exception: come l'RTS, ma per le eccezioni.

        end

Eseguendo questo listato prendete il valore dello Status Register nel momento
dell'eccezione, per cui alla fine dell'esecuzione in d0 ci sara' un valore,
solitamente $2000, che e' anche la prova che si stava eseguendo in exception,
dato che il bit 13 dello SR se settato indica il modo supervisore.

 (((
oO Oo
 \"/
  ~             5432109876543210
```

```
        ($2000=%0010000000000000)

NOTA: move.w SR,destinazione e' privilegiata solo dal 68010 in avanti, nel
68000 e' eseguibile anche in modo utente. Infatti chi la ha usata nelle
vecchie demo o giochi in modo utente, ha fatto si' che funzioni solo su 68000,
con lancio di bestemmie e accidenti per possessori di 68020+.
```

27.2 Lezione11b

```
; Lezione11b.s - Primo utilizzo della nuova startup2.s e di un interrupt.

        Section PrimoInterrupt,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$c020,$9a(a5)  ; INTENA - abilito interrupt "VERTB" del
                                ; livello 3 ($6c), quello che viene generato
                                ; una volta al fotogramma (alla linea $00).

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; lo interrompe per suonare la musica!).

        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB solamente.
*****************************************************************************
```

```
MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:      ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,COPER,VERTB
                                ; dato che il 680x0 non la cancella da solo!!!
        rte     ; uscita dall'int COPER/BLIT/VERTB

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.yellowcandy"

        end
```

Se non settassimo l'interrupt VERTB del livello 3 ($6c), questo listato si concluderebbe in un solo loop:

```
mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; lo interrompe per suonare la musica!).
```

Invece il processore lavora in "multitasking" bloccando il loop ogni volta che il pennello elettronico raggiunge la linea $00, eseguendo MT_MUSIC e ritornando ad eseguire lo sterile loop.
Anziche' questo vile loop di attesa del mouse, avremmo potuto mettere una routine di calcolo di un frattale, che poteva richiedere diversi secondi, durante i quali la musica avrebbe suonato in "contemporanea" e sincronizzata, senza disturbare il calcolo del frattale, rallentandolo solo il poco che serve a suonare la musica ogni fotogramma.

Da notare i 2 EQUATE all'inizio del programma, uno per l'accensione dei DMA, che ormai conosciamo, e quello nuovo:

```
WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)
```

Che "aspetta" un poco prima di prendere il controllo dell'hardware.
Per fare un calcolo del tempo atteso condiderate 50 come 1 secondo, essendo usato il Vblank, che va al "conquantesimo". Per cui 150 sono 3 secondi.
Se comunque il vostro programma e' un file abbastanza grosso e compattato,

a scompattare ci mettera' quel seconduccio o due che basta, per cui si puo' lasciare a un valore basso. Se invece salvaste il file non compresso, e lo faceste partire da dischetto, l'esecuzione partirebbe prima che la spia del drive si sia spenta, e una volta su 5 succede che all'uscita il dos e' andato in coma totale. Per evitare cio', calcolate sempre che tra scompattazione e tempo perso con il loop "waitdisk" il programma parta dopo 3 secondi almeno dalla fine del caricamento.

27.3 Lezione11c

```
; Lezione11c.s - Utilizzo di interrupts COPER e VERTB dell livello 3 ($6c).

        Section Interrupt,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$c030,$9a(a5)  ; INTENA - abilito interrupt "VERTB" e "COPER"
                                ; del livello 3 ($6c)

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************
```

```
MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:
        btst.b  #4,$dff01f      ; INTREQR - COPER azzerato?
        beq.s   NointCOPER      ; se si, non e' un int COPER!
        move.w  #$F00,$dff180   ; int COPER, allora COLOR0 = ROSSO
NointCOPER:
                 ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT e COPER
                                ; dato che il 680x0 non la cancella da solo!!!
        rte     ; uscita dall'int COPER/BLIT/VERTB

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU
        dc.w    $a007,$fffe     ; WAIT - attendi la linea $a0
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.yellowcandy"

        end
```

Questa volta abbiamo sfruttato anche l'interrupt del copper, detto COPER, utile per eseguire operazioni ad una certa linea video.
Da Copperlist infatti si puo' accedere anche al registro INTREQ ($dff09c), e in questo caso non facciamo altro che settare il bit 4, COPER, assieme al bit 15 Set/Clr.
In questo caso abbiamo messo solo un "MOVE.W #$f00,$dff180", che non e' un gran che di routine, ma considerate l'utilita' se le cose da fare sono molte, e non conviene perdere tempo a comparare il vertical blank con un loop del processore in modo user...

27.4 Lezione11d

```
; Lezione11d.s - Utilizzo di interrupts COPER e VERTB dell livello 3 ($6c).
```

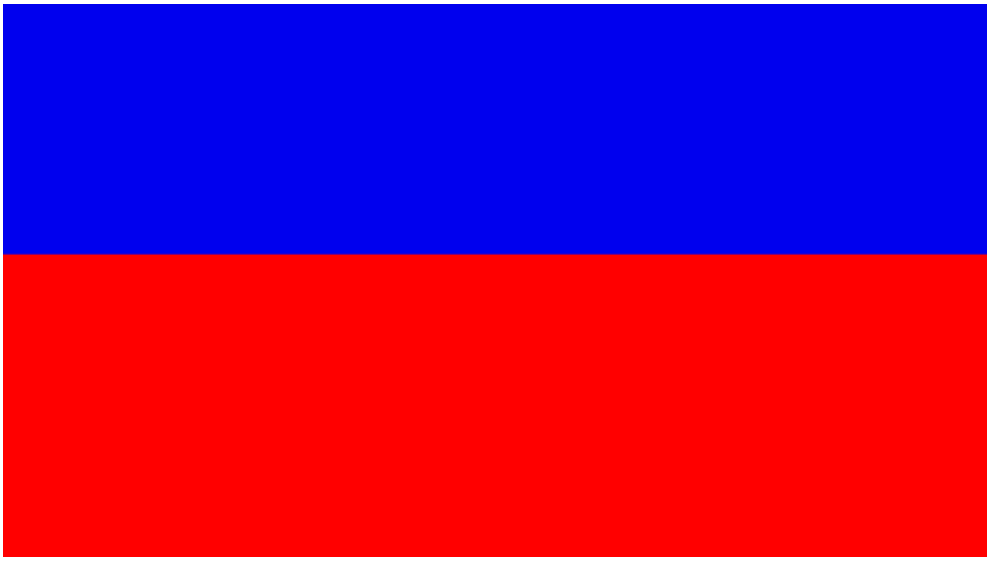


Figura 27.1: Lezione 11c

```
        Section Interrupt,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$c030,$9a(a5)  ; INTENA - abilito interrupt "VERTB" e "COPER"
                                ; del livello 3 ($6c)
```

```
mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:
        btst.b  #4,$dff01f      ; INTREQR - COPER azzerato?
        beq.s   NointCOPER      ; se si, non e' un int COPER!
        addq.b  #1,Attuale
        cmp.b   #6,Attuale
        bne.s   Vabene
        clr.b   attuale ; riparti da zero
VaBene:
        move.b  Attuale(PC),d0
        cmp.b   #1,d0
        beq.s   Col1
        cmp.b   #2,d0
        beq.s   Col2
        cmp.b   #3,d0
        beq.s   Col3
        cmp.b   #4,d0
        beq.s   Col4
        cmp.b   #5,d0
        beq.s   Col5
Col0:
        move.w  #$300,$dff180   ; COLOR0
        bra.s   Colorato
Col1:
        move.w  #$d00,$dff180   ; COLOR0
        bra.s   Colorato
Col2:
        move.w  #$f31,$dff180   ; COLOR0
        bra.s   Colorato
Col3:
        move.w  #$d00,$dff180   ; COLOR0
        bra.s   Colorato
Col4:
        move.w  #$a00,$dff180   ; COLOR0
        bra.s   Colorato
Col5:
        move.w  #$500,$dff180   ; COLOR0
Colorato:
NointCOPER:
                 ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,COPER,VERTB
                                ; dato che il 680x0 non la cancella da solo!!!
        rte     ; uscita dall'int COPER/BLIT/VERTB
```

```
Attuale:
        dc.w    0

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU
        dc.w    $a007,$fffe     ; WAIT - attendi la linea $a0
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $a207,$fffe     ; WAIT - attendi la linea $a2
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $a407,$fffe     ; WAIT - attendi la linea $a4
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $a607,$fffe     ; WAIT - attendi la linea $a6
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $a807,$fffe     ; WAIT - attendi la linea $a8
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $aa07,$fffe     ; WAIT - attendi la linea $aa
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.yellowcandy"

        end
```

In questo esempio vediamo come sia possibile chiamare l'interrupt a diverse linee, e come si possa ogni volta far eseguire una routine diversa, tramite l'uso di un contatore, la label "Attuale", che tiene il conto della routine da eseguire ad ogni chiamata. Se questo ordine viene cambiato, togliendo una routine, avverera' un "ciclare" delle routines. Provate, ad esempio, a fare questa modifica:

```
nointVERTB:
        btst.b  #4,$dff01f      ; INTREQR - COPER azzerato?
        beq.s   NointCOPER      ; se si, non e' un int COPER!
        addq.b  #1,Attuale
        cmp.b   #5,Attuale      ; ** MODIFICA ** -> 5, e non 6!!!!!!!!!
```

In questo modo vedrete uno scorrimento dei colori. Essendo pochi l'effetto e'

```
un po' troppo veloce, ma pensate all'utilita' se per ogni interrupt cambiaste
l'intera palette da 32 colori, e faceste anche qualcos'altro!
Senza contare il fatto che potete anche fare qualcosa nella routine "user",
che qua fa solo uno sterile ciclo in attesa della pressione del mouse.
```

Lezione11d2

```
; Lezione11d2.s - Utilizzo di interrupts COPER e VERTB dell livello 3 ($6c).
;                 Questa volta si fa ciclare la palette. Tasto destro del
;                 mouse per bloccare la routine temporaneamente.

        Section Interrupt,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper e bitplane DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

; Puntiamo la PIC

        MOVE.L  #PICTURE2,d0
        LEA     BPLPOINTERS2,A1
        MOVEQ   #5-1,D1                 ; num di bitplanes
POINTBT2:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #34*40,d0               ; lunghezza del bitplane
        addq.w  #8,a1
        dbra    d1,POINTBT2     ; Rifai D1 volte (D1=num do bitplanes)

; Puntiamo il nostro int di livello 3

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6
```

```
        move.w  #$c030,$9a(a5)  ; INTENA - abilito interrupt "VERTB" e "COPER"
                                ; del livello 3 ($6c)

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
        move.w  #$20,$dff09c    ; INTREQ - int eseguito, cancello la richiesta
                                ; dato che il 680x0 non la cancella da solo!!!
        rte     ; Uscita dall'int VERTB

nointVERTB:
        btst.b  #4,$dff01f      ; INTREQR - COPER azzerato?
        beq.s   NointCOPER      ; se si, non e' un int COPER!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   ColorCicla              ; Cicla i colori della pic
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack

NointCOPER:
                ;6543210
        move.w  #%1010000,$dff09c ; INTREQ - cancello richiesta BLIT e COPER
        rte     ; uscita dall'int COPER/BLIT


*****************************************************************************
*       Routine che "cicla" i colori di tutta la palette.                   *
*       Questa routine cicla i primi 15 colori separatamente dal secondo    *
*       secondo blocco di colori. Funziona come i "RANGE" del Dpaint.       *
*****************************************************************************

;       Il contatore "cont" serve a far aspettare 3 fotogrammi prima di
;       eseguire la routine cont. In pratica a "rallentare" l'esecuzione

cont:
        dc.w    0

ColorCicla:
        btst.b  #2,$dff016      ; Tasto destro del mouse premuto?
        beq.s   NonAncora       ; Se si, esci
        addq.b  #1,cont
        cmp.b   #3,cont         ; Agisci 1 volta ogni 3 fotogrammi solamente
        bne.s   NonAncora       ; Non siamo ancora al terzo? Esci!
        clr.b   cont            ; Siamo al terzo, azzera il contatore

; Roteazione all'indietro dei primi 15 colori
```

```
        lea     cols+2,a0       ; Indirizzo primo colore del primo gruppo
        move.w  (a0),d0         ; Salva il primo colore in d0
        moveq   #15-1,d7        ; 15 colori da "roteare" nel primo gruppo
cloop1:
        move.w  4(a0),(a0)      ; Copia il colore avanti in quello prima
        addq.w  #4,a0           ; salta alla prossimo col. da "indietreggiare"
        dbra    d7,cloop1       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema il primo colore salvato come ultimo.

; Roteazione in avanti dei secondi 15 colori

        lea     cole-2,a0       ; Indirizzo ultimo colore del secondo gruppo
        move.w  (a0),d0         ; Salva l'ultimo colore in d0
        moveq   #15-1,d7        ; Altri 15 colori da "roteare" separatamente
cloop2:
        move.w  -4(a0),(a0)     ; Copia il colore indietro in quello dopo
        subq.w  #4,a0           ; salta al precedente col. da "avanzare"
        dbra    d7,cloop2       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema l'ultimo colore salvato come primo
NonAncora:
        rts


*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $b807,$fffe     ; WAIT - attendi la linea $b8
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sui 32 colori della palette.

        dc.w    $b907,$fffe     ; WAIT - attendi linea $b9
BPLPOINTERS2:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "

        dc.w    $100,%0101001000000000  ; BPLCON0 - 5 bitplanes LOWRES

; La palette, che sara' "ruotata" in 2 gruppi di 16 colori.

cols:
        dc.w $180,$040,$182,$050,$184,$060,$186,$080    ; tono verde
```

```
        dc.w $188,$090,$18a,$0b0,$18c,$0c0,$18e,$0e0
        dc.w $190,$0f0,$192,$0d0,$194,$0c0,$196,$0a0
        dc.w $198,$090,$19a,$070,$19c,$060,$19e,$040

        dc.w $1a0,$029,$1a2,$02a,$1a4,$13b,$1a6,$24b    ; tono blu
        dc.w $1a8,$35c,$1aa,$36d,$1ac,$57e,$1ae,$68f
        dc.w $1b0,$79f,$1b2,$68f,$1b4,$58e,$1b6,$37e
        dc.w $1b8,$26d,$1ba,$15d,$1bc,$04c,$1be,$04c
cole:

        dc.w    $da07,$fffe     ; WAIT - attendi la linea $da
        dc.w    $100,$200       ; BPLCON0 - disabilita i bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


*****************************************************************************
;               DISEGNO 320*34 a 5 bitplanes (32 colori)
*****************************************************************************

PICTURE2:
        INCBIN  "pic320*34*5.raw"

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.fuck the bass"

        end
```

In questo esempio cambiato la palette proprio una linea prima del disegno.
Infatti basta cambiarlo una linea prima!
Nel frattempo col processore si possono svolgere compiti vari, ma saremo
sicuri che alla linea $b9 i colori sono cambiati ogni volta.
Altra cosa da notare e' che nonostante l'interrupt avvenga ogni fotogramma,
tramite un "contatore" e' possibile far eseguire la routine una volta ogni
3 fotogrammi. Dunque abbiamo visto che e' possibile mettere piu' routines
e piu' interrupt nella stessa copperlist, a linee diverse, in Lezione11d.s,
basta fare in modo che ogni volta sia eseguita la routine per quella linea.
Ora vediamo che si puo' far eseguire qualcuna di queste routine una volta
ogni X frames, per cui si puo' fare ogni cosa!
Ricordatevi pero' che ogni interrupt porta via un poco di tempo per i salti
che devono essere fatti.

Una nota: i 2 numeri sono della bbs Fidonet AmigaLink di Grosseto, infatti
questo e' un "pezzo" della piccola demo che ho fatto al sysop di questa bbs.
Sono segnato come "Fabio Ciucci", ma raramente chiamo, per motivi tutti
di bolletta galattica. Fino a che non ci sara' l'accesso gratuito in tutte le
citta' ad Internet per i coder sara' dura scambiare via modem. Meglio la posta!

27.5 Lezione11e

```
; Lezione11e.s - Utilizzo di interrupts COPER e VERTB dell livello 3 ($6c).
;                In questo caso ridefiniamo tutti gli interrupt, giusto
```

```
;               per rendere l'idea di come si fa.

        Section Interrupt,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR

        MOVE.L  #NOINT1,$64(A0)         ; Interrupt "vuoto"
        MOVE.L  #NOINT2,$68(A0)         ; int vuoto
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        MOVE.L  #NOINT4,$70(A0)         ; int vuoto
        MOVE.L  #NOINT5,$74(A0)         ; " "
        MOVE.L  #NOINT6,$78(A0)         ; " "

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

               ; 5432109876543210
        move.w  #%1111111111111111,$9a(a5)    ; INTENA - abilito TUTTI gli
                                              ; interrupt!

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci


*****************************************************************************
*       ROUTINE IN INTERRUPT $64 (livello 1)
*****************************************************************************

;02     SOFT    1 ($64) Riservato agli interrupt inizializzati via software.
;01     DSKBLK  1 ($64) Fine del trasferimento di un blocco dati dal disco.
;00     TBE     1 ($64) Buffer UART di trasmissione della porta seriale VUOTO.
```

```
NOINT1: ; $64
        btst.b  #0,$dff01f      ; INTREQR - TBE?
        beq.w   NoTBE
        ; tbe routines
NoTBE:
        btst.b  #1,$dff01f      ; INTREQR - DSKBLK?
        beq.w   NoDSKBLK
        ; DSKBLK routines
NoDSKBLK:
        btst.b  #2,$dff01f      ; INTREQR - SOFT?
        beq.w   NoSOFT
        ; SOFT routines
NoSOFT:
                ; 210
        move.w  #%111,$dff09c   ; INTREQ - soft,dskblk,serial port tbe
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $68 (livello 2)
*****************************************************************************

;03     PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2

NOINT2: ; $68
        btst.b  #3,$dff01f      ; INTREQR - PORTS?
        beq.w   NoPORTS
        ; routines PORTS
NoPORTS:
        move.l  d0,-(sp)        ; salva d0
        move.b  $bfed01,d0      ; CIAA icr - e' un interrupt della tastiera?
        and.b   #$8,d0
        beq.w   NoTastiera
        ; Routines per la lettura della tastiera
NoTastiera:
        move.l  (sp)+,d0        ; ripristina d0
                ; 3210
        move.w  #%1000,$dff09c  ; INTREQ - ports
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.      *
*****************************************************************************

;06     BLIT    3 ($6c) Se il blitter ha finito una blittata si setta ad 1
;05     VERTB   3 ($6c) Generato ogni volta che il pennello elettronico e'
;                       alla linea 00, ossia ad ogni inizio di vertical blank.
;04     COPER   3 ($6c) Si puo' settare col copper per generarlo ad una certa
;                       linea video. Basta richiederlo dopo un certo WAIT.

MioInt6c:
        btst.b  #6,$dff01f      ; INTREQR - BLIT?
        beq.w   NoBLIT
        ; routines BLIT
NoBLIT:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:
        btst.b  #4,$dff01f      ; INTREQR - COPER azzerato?
```

```
        beq.s   NointCOPER      ; se si, non e' un int COPER!
        move.w  #$F00,$dff180   ; int COPER, allora COLOR0 = ROSSO
NointCOPER:
                ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,VERTB e COPER
        rte     ; uscita dall'int COPER/BLIT/VERTB

*****************************************************************************
*       ROUTINE IN INTERRUPT $70 (livello 4)
*****************************************************************************

;10     AUD3    4 ($70) Lettura di un blocco di dati del can. audio 3 finita.
;09     AUD2    4 ($70) Lettura di un blocco di dati del can. audio 2 finita.
;08     AUD1    4 ($70) Lettura di un blocco di dati del can. audio 1 finita.
;07     AUD0    4 ($70) Lettura di un blocco di dati del can. audio 0 finita.

NOINT4: ; $70
        BTST.b  #7,$dff01f      ; INTREQR - AUD0?
        BEQ.W   NoAUD0
        ; routines aud0
NoAUD0:
        BTST.b  #8-7,$dff01e    ; INTREQR - AUD1? nota: $dff01e e non $dff01f
                                ;                 perche' il bit e' >7!
        BEQ.W   NoAUD1
        ; routines aud1
NoAUD1:
        BTST.b  #9-7,$dff01e    ; INTREQR - AUD2?
        Beq.W   NoAUD2
        ; routines aud2
NoAUD2:
        BTST.b  #10-7,$dff01e   ; INTREQR - AUD3?
        Beq.W   NoAUD3
        ; routines aud3
NoAUD3:
                ; 09876543210
        MOVE.W  #%11110000000,$DFF09C   ; aud0,aud1,aud2,aud3
        RTE

*****************************************************************************
*       ROUTINE IN INTERRUPT $74 (livello 5)
*****************************************************************************

;12     DSKSYN  5 ($74) Generato se il registro DSKSYNC corrisponde ai dati
;                       letti dal disco nel drive.Serve per i loader hardware.
;11     RBF     5 ($74) Buffer UART di ricezione della porta seriale PIENO.


NOINT5: ; $74
        BTST.b  #12-7,$dff01e   ; INTREQR - DSKSYN?
        BEQ.W   NoDSKSYN
        ; routines dsksyn
NoDSKSYN:
        BTST.b  #11-7,$dff01e   ; INTREQR - RBF?
        BEQ.W   NoRBF
        ; routines rbf
NoRBF:
                ; 2109876543210
        MOVE.W  #%1100000000000,$DFF09C ; serial port rbf, dsksyn
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $78 (livello 6)                                *
```

```
*****************************************************************************

;14     INTEN   6 ($78)
;13     EXTER   6 ($78) Interrupt esterno, connesso alla linea INT6 + TOD CIAB

NOINT6: ; $78
        tst.b   $bfdd00         ; CIAB icr - resetta interrupt timer
        BTST.b  #14-7,$dff01e   ; INTREQR - INTEN?
        BEQ.W   NoINTEN
        ; routines inten
NoINTEN:
        BTST.b  #13-7,$dff01e   ; INTREQR - EXTER?
        BEQ.W   NoEXTER
        ; routines exter
NoEXTER:
                ; 432109876543210
        MOVE.W  #%1100000000000000,$DFF09C ; INTREQ - external int + ciab
        rte

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU
        dc.w    $a007,$fffe     ; WAIT - attendi la linea $a0
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.fairlight"

        end

Abbiamo ridefinito tutti gli interrupt. Questo puo' essere uno schema di
partenza per farsi un "sistema operativo", ma ve lo sconsiglio!
```

27.6 Lezione11f

```
; Lezione11f.s - Utilizzo di interrupts COPER e VERTB dell livello 3 ($6c).
;                In questo caso ridefiniamo tutti gli interrupt, giusto
;                per rendere l'idea di come si fa.
;                La differenza con Lezione11e.s e' "formale", infatti si usa
;                per gli interrupt lo standard della ROM Amiga. Se volete
;                seguire proprio l'etichetta, fate come in questo esempio.

        Section Interrupt,CODE
```

```
;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR

        MOVE.L  #NOINT1,$64(A0)         ; Interrupt "vuoto"
        MOVE.L  #NOINT2,$68(A0)         ; int vuoto
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        MOVE.L  #NOINT4,$70(A0)         ; int vuoto
        MOVE.L  #NOINT5,$74(A0)         ; " "
        MOVE.L  #NOINT6,$78(A0)         ; " "

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

                ; 5432109876543210
        move.w  #%1111111111111111,$9a(a5)    ; INTENA - abilito TUTTI gli
                                              ; interrupt!

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci


*****************************************************************************
*       ROUTINE IN INTERRUPT $64 (livello 1)
*****************************************************************************

;02     SOFT    1 ($64) Riservato agli interrupt inizializzati via software.
;01     DSKBLK  1 ($64) Fine del trasferimento di un blocco dati dal disco.
;00     TBE     1 ($64) Buffer UART di trasmissione della porta seriale VUOTO.

NOINT1: ; $64
        movem.l d0-d7/a0-a6,-(SP)
```

```
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
        BEQ.s   NoInts1         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        btst.l  #0,d1           ; TBE?
        beq.w   NoTBE
        ; tbe routines
NoTBE:
        btst.l  #1,d1           ; DSKBLK?
        beq.w   NoDSKBLK
        ; DSKBLK routines
NoDSKBLK:
        btst.l  #2,d1           ; INTREQR - SOFT?
        beq.w   NoSOFT
        ; SOFT routines
NoSOFT:
NoInts1:        ; 210
        move.w  #%111,$dff09c   ; INTREQ - soft,dskblk,serial port tbe
        movem.l (SP)+,d0-d7/a0-a6
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $68 (livello 2)
*****************************************************************************

;03     PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2

NOINT2: ; $68
        movem.l d0-d7/a0-a6,-(SP)
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
        BEQ.s   NoInts2         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.

        btst.l  #3,d1           ; INTREQR - PORTS?
        beq.w   NoPORTS
        ; routines PORTS
NoPORTS:
        move.l  d0,-(sp)        ; salva d0
        move.b  $bfed01,d0      ; CIAA icr - e' un interrupt della tastiera?
        and.b   #$8,d0
        beq.w   NoTastiera
        ; Routines per la lettura della tastiera
NoTastiera:
        move.l  (sp)+,d0        ; ripristina d0
NoInts2:        ; 3210
        move.w  #%1000,$dff09c  ; INTREQ - ports
        movem.l (SP)+,d0-d7/a0-a6
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.      *
*****************************************************************************
```

```
;        _.--._     _
;       |   _ .|   (_)
;       |   \__|   ||
;       |______|   ||
;       .-‘--’-.   ||
;       | | |  |\__l|
;       |_| |__|__|_))
;        ||_| |    ||
;        |(_) |
;        |    |
;        |____|__
;        |______/g®m


;06     BLIT    3 ($6c) Se il blitter ha finito una blittata si setta ad 1
;05     VERTB   3 ($6c) Generato ogni volta che il pennello elettronico e'
;                       alla linea 00, ossia ad ogni inizio di vertical blank.
;04     COPER   3 ($6c) Si puo' settare col copper per generarlo ad una certa
;                       linea video. Basta richiederlo dopo un certo WAIT.

MioInt6c:
        movem.l d0-d7/a0-a6,-(SP)
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
        BEQ.s   NoInts3         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        btst.l  #6,d1           ; INTREQR - BLIT?
        beq.w   NoBLIT
        ; routines BLIT
NoBLIT:
        btst.l  #5,d1           ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:
        btst.l  #4,d1           ; INTREQR - COPER azzerato?
        beq.s   NointCOPER      ; se si, non e' un int COPER!
        move.w  #$F00,$dff180   ; int COPER, allora COLOR0 = ROSSO
NointCOPER:
NoInts3:         ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,VERTB,COPER
        movem.l (SP)+,d0-d7/a0-a6
        rte     ; uscita dall'int COPER/BLIT

*****************************************************************************
*       ROUTINE IN INTERRUPT $70 (livello 4)
*****************************************************************************

;10     AUD3    4 ($70) Lettura di un blocco di dati del can. audio 3 finita.
;09     AUD2    4 ($70) Lettura di un blocco di dati del can. audio 2 finita.
;08     AUD1    4 ($70) Lettura di un blocco di dati del can. audio 1 finita.
;07     AUD0    4 ($70) Lettura di un blocco di dati del can. audio 0 finita.

NOINT4: ; $70
        movem.l d0-d7/a0-a6,-(SP)
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
```

```
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
        BEQ.s   NoInts4         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        BTST.l  #7,d1           ; INTREQR - AUD0?
        BEQ.W   NoAUD0
        ; routines aud0
NoAUD0:
        BTST.l  #8,d1           ; INTREQR - AUD1?
        BEQ.W   NoAUD1
        ; routines aud1
NoAUD1:
        BTST.l  #9,d1           ; INTREQR - AUD2?
        Beq.W   NoAUD2
        ; routines aud2
NoAUD2:
        BTST.l  #10,d1          ; INTREQR - AUD3?
        Beq.W   NoAUD3
        ; routines aud3
NoAUD3:
NoInts4:        ; 09876543210
        MOVE.W  #%11110000000,$DFF09C   ; aud0,aud1,aud2,aud3
        movem.l (SP)+,d0-d7/a0-a6
        RTE

*****************************************************************************
*       ROUTINE IN INTERRUPT $74 (livello 5)
*****************************************************************************

;12     DSKSYN  5 ($74) Generato se il registro DSKSYNC corrisponde ai dati
;                       letti dal disco nel drive.Serve per i loader hardware.
;11     RBF     5 ($74) Buffer UART di ricezione della porta seriale PIENO.


NOINT5: ; $74
        movem.l d0-d7/a0-a6,-(SP)
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
        BEQ.s   NoInts5         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        BTST.l  #12,d1          ; INTREQR - DSKSYN?
        BEQ.W   NoDSKSYN
        ; routines dsksyn
NoDSKSYN:
        BTST.l  #11,d1          ; INTREQR - RBF?
        BEQ.W   NoRBF
        ; routines rbf
NoRBF:
NoInts5:        ; 2109876543210
        MOVE.W  #%1100000000000,$DFF09C ; serial port rbf, dsksyn
        movem.l (SP)+,d0-d7/a0-a6
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $78 (livello 6)                                *
*****************************************************************************
```

```
;14     INTEN   6 ($78)
;13     EXTER   6 ($78) Interrupt esterno, connesso alla linea INT6 + TOD CIAB

NOINT6: ; $78
        movem.l d0-d7/a0-a6,-(SP)
        tst.b   $bfdd00         ; CIAB icr - resetta interrupt timer
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
        BEQ.s   NoInts6         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        BTST.l  #14,d1          ; INTREQR - INTEN?
        BEQ.W   NoINTEN
        ; routines inten
NoINTEN:
        BTST.l  #13,d1          ; INTREQR - EXTER?
        BEQ.W   NoEXTER
        ; routines exter
NoEXTER:
NoInts6:        ; 432109876543210
        MOVE.W  #%110000000000000,$DFF09C ; INTREQ - external int + ciab
        movem.l (SP)+,d0-d7/a0-a6
        rte

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU
        dc.w    $a007,$fffe     ; WAIT - attendi la linea $a0
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, il
                                ; quale fa agire sul color0 con un "MOVE.W".
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.yellowcandy"

        end
```

COme potete notare, in questa "versione" si usano tutti gli accorgimenti che usano gli interrupt del sistema operativo:

```
        LEA     $DFF000,A0      ; custom in A0
        MOVE.W  $1C(A0),D1      ; INTENAR in d1
        BTST.l  #14,D1          ; Bit Master di abilitazione azzerato?
```

```
        BEQ.s   NoInts1         ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D1      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        btst.l  #0,d1           ; TBE?
        ...
```

In pratica si fa un'ulteriore verifica della validita' dell'interrupt, controllando se il bit settato in INTREQR e' settato anche in INTENAR, ossia se e' abilitato. In realta' se si disabilita un interrupt con il registro apposito, ossia INTENA ($dff09a), non dovrebbe continuare a funzionare. Comunque puo' essere che l'hardware non sia perfetto. Se trovate delle strane incompatibilita' nel vostro interrupt, fate anche questo controllo, chissa' che non vengano eseguiti interrupt "disabilitati"!

27.7 Lezione11g1

```
; Lezione11g1.s -  Uso della caratteristica del copper di richiedere 8 pixel
;                  orizzontali per eseguire un suo "MOVE".

        Section HorizCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        BSR.W   MAKE_IT         ; Prepara la copperlist

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA
MOUSE:
        BTST    #$06,$BFE001    ; Aspetta la presisone del mouse
        BNE.S   MOUSE
        RTS

*****************************************************************************
*   Questa routine crea una copperlist con 40 registri COLOR0 per       *
*   Linea, per cui, dato che ogni move della copperlist impiega 8       *
*   pixel (lowres) di tempo per essere eseguita, il color0 viene        *
*   cambiato 40 volte ORIZZONTALMENTE a scatti di 8 pixel lowres        *
*****************************************************************************

LINSTART        EQU     $A041fffe       ; Cambiare "$a0" per iniziare ad un
                                        ; altra linea verticale.
```

```
LINUM           EQU     25              ; Numero di linee da fare.

MAKE_IT:
        lea     CopBuf,a1
        move.l  #LINSTART,d0    ; Primo "wait"
        move.w  #LINUM-1,d1     ; Numero di linee da fare
colcon1:
        lea     cols(pc),a0     ; Indirizzo tabella con i colori in a0
        move.w  #39-1,d2        ; 39 colori per linea
        move.l  d0,(a1)+        ; Metti il WAIT in copperlist
colcon2:
        move.w  #$0180,(a1)+    ; Metti il registro COLOR0
        move.w  (a0)+,(a1)+     ; Metti il valore del COLOR0 (dalla tabella)
        dbra    d2,colcon2      ; Esegui tutta una linea
        add.l   #$01000000,d0   ; Fai "waitare" alla linea sotto
        dbra    d1,colcon1      ; ripeti per il numero di linee da fare
        rts


;       Tabella con i 39 colori di una linea orizzontale.

cols:
        dc.w    $000,$111,$222,$333,$444,$555,$666,$777
        dc.w    $888,$999,$aaa,$bbb,$ccc,$ddd,$eee,$fff
        dc.w    $fff,$fff,$fff,$fff,$fff,$fff,$fff,$fff
        dc.w    $eee,$ddd,$ccc,$bbb,$aaa,$999,$888,$777
        dc.w    $666,$555,$444,$333,$222,$111,$000

*****************************************************************************

        section coppa,data_C

COPLIST:
        DC.W    $100,$200       ; BplCon0 - no bitplanes
        DC.W    $180,$003       ; Color0 - blu
CopBuf:
        dcb.w   80*LINUM,0      ; Spazio dove sara' creata la copperlist.

        DC.W    $180,$003       ; Color0 - blu
        dc.w    $ffff,$fffe     ; Fine copperlist

        END
```

Questo listato dimostra come mettendo una fila di COLOR0 (o di qualsiasi altro MOVE del WAIT), ci vuole un certo tempo per eseguirne ciascuno, e precisamente 8 pixel lowres. Infatti se si mette la risoluzione hires questo non cambia, e si puo' parlare di "16" pixel hires... ma e' inutile, se volete potete misurare la larghezza di uno "scatto" orizzontale con un righello e noterete che e' sempre quella. Oltre a essere un fatto utile per effetti come il PLASMI o quello visto in questo esempio, e' una limitazione, nel senso che se si vuol cambiare tutta la palette ad ogni riga ci vuole "un certo tempo" e questa non cambierebbe completamente che a meta' linea o addirittura alla linea sotto.

Lezione11g2

```
; Lezione11g2.s -  Uso della caratteristica del copper di richiedere 8 pixel
;                  orizzontali per eseguire un suo "MOVE".

        Section coppuz,CODE
```

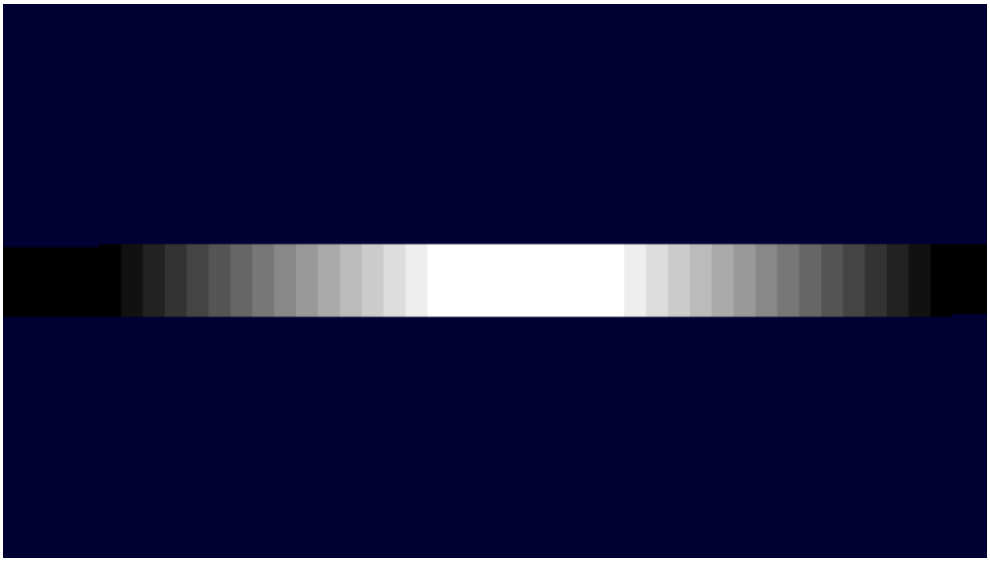


Figura 27.2: Lezione 11g1

```
;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        BSR.W   MAKE_IT         ; Prepara la copperlist

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA
MOUSE:
        BTST    #$06,$BFE001    ; Aspetta la presisone del mouse
        BNE.S   MOUSE
        RTS

*****************************************************************************
*   Questa routine crea una copperlist con 52 registri COLOR0 per       *
*   Linea, per cui, dato che ogni move della copperlist impiega 8       *
*   pixel (lowres) di tempo per essere eseguita, il color0 viene        *
*   cambiato 52 volte ORIZZONTALMENTE a scatti di 8 pixel lowres        *
```

```
*****************************************************************************

LINSTART        EQU     $8021fffe       ; Cambiare "$80" per iniziare ad un
                                        ; altra linea verticale.
LINUM           EQU     25*3            ; Numero di linee da fare.

MAKE_IT:
        lea     CopBuf,a1       ; Indirizzo spazio in copperlist
        move.l  #LINSTART,d0    ; Primo "wait"
        move.w  #LINUM-1,d1     ; Numero di linee da fare
        move.w  #$180,d3        ; Word per il registro color0 in coplist
        move.l  #$01000000,d4   ; Valore da "addare" al wait per farlo waitare
                                ; alla linea successiva.
colcon1:
        lea     cols(pc),a0     ; Indirizzo tabella con i colori in a0
        move.w  #52-1,d2        ; 52 colori per linea
        move.l  d0,(a1)+        ; Metti il WAIT in copperlist
colcon2:
        move.w  d3,(a1)+        ; Metti il registro COLOR0 ($180)
        move.w  (a0)+,(a1)+     ; Metti il valore del COLOR0 (dalla tabella)
        dbra    d2,colcon2      ; Esegui tutta una linea
        add.l   d4,d0           ; Fai "waitare" alla linea sotto (+$01000000)
        dbra    d1,colcon1      ; ripeti per il numero di linee da fare
        rts


;       Tabella con i 52 colori di una linea orizzontale.

cols:
        dc.w    $26F,$27E,$28D,$29C,$2AB,$2BA,$2C9,$2D8,$2E7,$2F6
        dc.w    $4E7,$6D8,$8C9,$ABA,$CAA,$D9A,$E8A,$F7A,$F6B,$F5C
        dc.w    $D6D,$B6E,$96F,$76F,$56F,$36F,$26F,$27E,$28D,$29C
        dc.w    $2AB,$2BA,$2C9,$2D8,$2E7,$2F6,$4E7,$6D8,$8C9,$ABA
        dc.w    $CAA,$D9A,$E8A,$F7A,$F6B,$F5C,$D6D,$B6E,$96F,$76F
        dc.w    $56F,$36F

*****************************************************************************

        section coppa,data_C

COPLIST:
        DC.W    $100,$200       ; BplCon0 - no bitplanes
        DC.W    $180,$003       ; Color0 - blu
CopBuf:
        dcb.w   (52*2)*LINUM+(2*linum),0        ; Spazio per la copperlist.
        DC.W    $180,$003       ; Color0 - blu
        dc.w    $ffff,$fffe     ; Fine copperlist

        END
```

In questo caso abbiamo reso piu' "colorato" l'effetto, niente di speciale.

Lezione11g3

```
; Lezione11g3.s -  Uso della caratteristica del copper di richiedere 8 pixel
;                  orizzontali per eseguire un suo "MOVE".

        Section coppuz,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        BSR.W   MAKE_IT         ; Prepara la copperlist

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA
MOUSE:
        BTST    #$06,$BFE001    ; Aspetta la presisone del mouse
        BNE.S   MOUSE
        RTS

*****************************************************************************
*   Questa routine crea una copperlist con 52 registri COLOR0 per       *
*   Linea, per cui, dato che ogni move della copperlist impiega 8       *
*   pixel (lowres) di tempo per essere eseguita, il color0 viene        *
*   cambiato 52 volte ORIZZONTALMENTE a scatti di 8 pixel lowres        *
*****************************************************************************

LINSTART        EQU     $8021fffe       ; Cambiare "$80" per iniziare ad un
                                        ; altra linea verticale.
LINUM           EQU     80              ; Numero di linee da fare.

MAKE_IT:
        lea     cols(pc),a0     ; Indirizzo tabella con i colori in a0
        lea     CopBuf,a1       ; Indirizzo spazio in copperlist
        move.l  #LINSTART,d0    ; Primo "wait"
        move.w  #LINUM-1,d1     ; Numero di linee da fare
        move.w  #$180,d3        ; Word per il registro color0 in coplist
        move.l  #$01000000,d4   ; Valore da "addare" al wait per farlo waitare
                                ; alla linea successiva.
        moveq   #9,d6           ; e metti il contatore a 9
colcon1:
        move.w  #52-1,d2        ; 52 colori per linea
        move.l  d0,(a1)+        ; Metti il WAIT in copperlist
colcon2:
        move.w  d3,(a1)+        ; Metti il registro COLOR0 ($180)
        move.w  (a0)+,(a1)+     ; Metti il valore del COLOR0 (dalla tabella)
        dbra    d2,colcon2      ; Esegui tutta una linea
        add.l   d4,d0           ; Fai "waitare" alla linea sotto (+$01000000)
        subq.b  #1,d6           ; segna che abbiamo fatto una linea
        bne.s   NonRipartire    ; se ne abbiamo fatte 8, d6=0, allora occorre
                                ; ripartire dal primo colore nella tabella.
        lea     cols(pc),a0     ; tab colori in a0 - riparti col colori.
        moveq   #9,d6           ; e metti il contatore a 8
NonRipartire:
        dbra    d1,colcon1      ; ripeti per il numero di linee da fare
```

```
        rts


;       Tabella con i 52*9 colori di una linea orizzontale.

cols:
        dc.w    $26F,$27E,$28D,$29C,$2AB,$2BA,$2C9,$2D8,$2E7,$2F6
        dc.w    $4E7,$6D8,$8C9,$ABA,$CAA,$D9A,$E8A,$F7A,$F6C,$F5C
        dc.w    $D6D,$B6E,$96F,$76F,$56F,$36F,$26F,$27E,$28D,$29C
        dc.w    $2AB,$2BA,$2C9,$2D8,$2E7,$2F6,$4E7,$6D8,$8C9,$ABA
        dc.w    $CAA,$D9A,$E8A,$F7A,$F6B,$F5C,$D6D,$B6E,$96F,$76F
        dc.w    $56F,$36F,$36F,$37E,$38D,$39C,$3AB,$3BA,$3C9,$3D8
        dc.w    $3E7,$3F6,$4E7,$7D8,$9C9,$BBA,$DAA,$E9A,$F8A,$F7A
        dc.w    $F6C,$F5C,$E6D,$C6E,$A6F,$86F,$66F,$46F,$36F,$37E
        dc.w    $38D,$39C,$3AB,$3BA,$3C9,$3D8,$3E7,$3F6,$5E7,$7D8
        dc.w    $9C9,$BBA,$DAA,$E9A,$F8A,$F7A,$F6B,$F5C,$E6D,$C6E
        dc.w    $A6F,$86F,$46F,$46F,$36E,$37D,$38C,$39B,$3AA,$3B9
        dc.w    $3C8,$3D7,$3E6,$3F5,$4E6,$7D7,$9C8,$BB9,$DA9,$E99
        dc.w    $F89,$F79,$F6B,$F5B,$E6C,$C6D,$A6E,$86E,$66E,$46E
        dc.w    $36E,$37D,$38C,$39B,$3AA,$3B9,$3C8,$3D7,$3E6,$3F5
        dc.w    $5E6,$7D7,$9C8,$BB9,$DA9,$E99,$F89,$F79,$F6A,$F5B
        dc.w    $E6C,$C6E,$A6E,$86E,$46E,$46E,$46E,$47D,$48C,$49B
        dc.w    $4AA,$4B9,$4C8,$4D7,$4E6,$4F5,$5E6,$8D7,$AC8,$CB9
        dc.w    $EA9,$F99,$F89,$F79,$F6B,$F5B,$F6C,$D6D,$B6E,$96E
        dc.w    $76E,$56E,$46E,$47D,$48C,$49B,$4AA,$4B9,$4C8,$4D7
        dc.w    $4E6,$4F5,$6E6,$8D7,$AC8,$CB9,$EA9,$F99,$F89,$F79
        dc.w    $F6A,$F5B,$F6C,$D6E,$B6E,$96E,$56E,$56E,$45E,$46D
        dc.w    $47C,$48B,$49A,$4A9,$4B8,$4C7,$4D6,$4E5,$5D6,$8C7
        dc.w    $AB8,$CA9,$E99,$F89,$F79,$F69,$F5B,$F4B,$F5C,$D5D
        dc.w    $B5E,$95E,$75E,$55E,$45E,$46D,$47C,$48B,$49A,$4A9
        dc.w    $4B8,$4C7,$4D6,$4E5,$6D6,$8C7,$AB8,$CA9,$E99,$F89
        dc.w    $F79,$F69,$F5A,$F4B,$F5C,$D5E,$B5E,$95E,$55E,$55E
        dc.w    $44D,$45C,$46B,$47A,$489,$498,$4A7,$4B6,$4C5,$4D4
        dc.w    $5C5,$8B6,$AA7,$C98,$E88,$F78,$F68,$F68,$F59,$F4A
        dc.w    $F4B,$D4C,$B4D,$94D,$74D,$54D,$44D,$45C,$46B,$47A
        dc.w    $489,$498,$4A7,$4B6,$4C5,$4D4,$6C5,$8B6,$AA7,$C98
        dc.w    $E88,$F78,$F68,$F58,$F49,$F3A,$F4B,$D4D,$B4D,$94D
        dc.w    $54D,$54D,$44C,$45B,$46A,$479,$488,$499,$4A6,$4B5
        dc.w    $4C4,$4D3,$5C4,$8B5,$AA6,$C97,$E87,$F77,$F67,$F67
        dc.w    $F58,$F49,$F4C,$D4B,$B4C,$94C,$74C,$54C,$44C,$45B
        dc.w    $46A,$479,$488,$497,$4A6,$4B5,$4C4,$4D3,$6C4,$8B5
        dc.w    $AA6,$C97,$E87,$F77,$F67,$F57,$F48,$F39,$F4A,$D4C
        dc.w    $B4C,$94C,$54C,$54C,$44B,$45A,$469,$478,$487,$498
        dc.w    $4A5,$4B4,$4C3,$4D2,$5C3,$8B4,$AA5,$C96,$E86,$F76
        dc.w    $F66,$F66,$F57,$F48,$F4B,$D4A,$B4B,$94B,$74B,$54B
        dc.w    $44B,$45A,$469,$478,$487,$496,$4A5,$4B4,$4C3,$4D2
        dc.w    $6C3,$8B4,$AA5,$C96,$E86,$F76,$F66,$F56,$F47,$F38
        dc.w    $F49,$D4B,$B4B,$94B,$54B,$54B,$44A,$459,$468,$477
        dc.w    $486,$497,$4A4,$4B3,$4C2,$4D1,$5C2,$8B3,$AA4,$C95
        dc.w    $E85,$F75,$F65,$F65,$F56,$F47,$F4A,$D49,$B4A,$94A
        dc.w    $74A,$54A,$44A,$459,$468,$477,$486,$495,$4A4,$4B3
        dc.w    $4C2,$4D1,$6C2,$8B3,$AA4,$C95,$E85,$F75,$F65,$F55
        dc.w    $F46,$F37,$F48,$D4A,$B4A,$94A,$54A,$54A

*****************************************************************************

        section coppa,data_C

COPLIST:
        DC.W    $100,$200       ; BplCon0 - no bitplanes
        DC.W    $180,$003       ; Color0 - blu
CopBuf:
```

```
        dcb.w   (52*2)*LINUM+(2*linum),0        ; Spazio per la copperlist.

        DC.W    $180,$003       ; Color0 - blu
        dc.w    $ffff,$fffe     ; Fine copperlist

        END
```

Sempre piu' colorato, ma in sostanza non e' cambiato niente da Lezione11g1.s

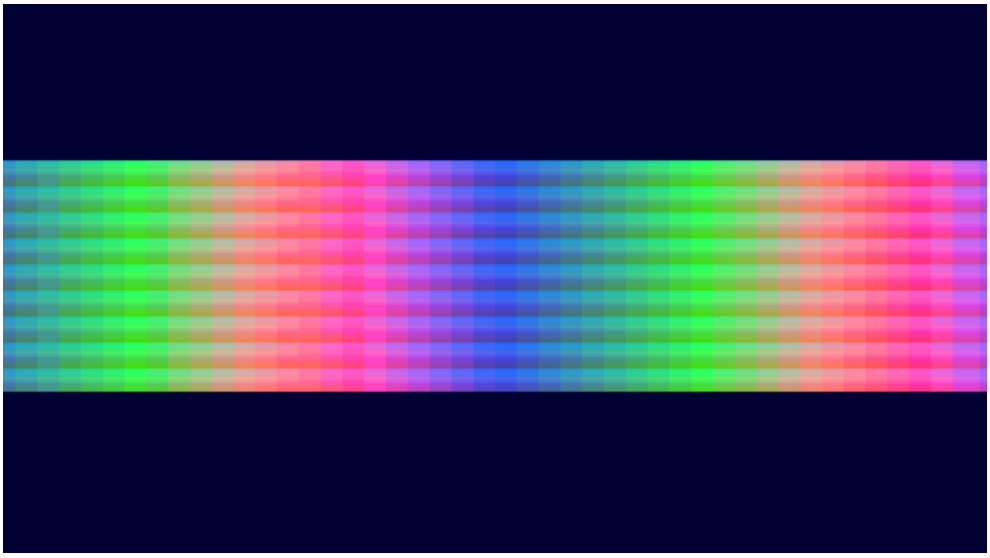


Figura 27.3: Lezione 11g3

Lezione11g4

```
; Lezione11g4.s - Effetto di scorrimento orizzontale dei colori col Copper

        SECTION Supercar,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
```

```
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        btst    #2,$dff016      ; tasto destro premuto?
        beq.s   Mouse2          ; se si non eseguire Linecop

        bsr.s   LineCop         ; Effetto "supercar"

mouse2:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

; Questa routine immette i colori ciclicamente dalla tabella alle due linee
; formate da 54 "dc.w $1800000". L'effetto e' possibile in quanto ogni volta
; che il copper legge una istruzione, formata da 2 words, il tempo necessario
; per arrivare a leggere quella seguente corrisponde a 4 pixel per word, ossia
; 8 pixel per ogni istruzione completa. Facendo un calcolo, in uno schermo
; largo 320 pixel si puo' cambiare il colore orizzontalmente 40 volte, infatti
; 320/8=40. In questo caso si parte dalla posizione verticale overscan,
; ossia fuori dai bordi del monitor (il wait e' dc.w $2901,$FFFE), e si arriva
; fuori al bordo destro dall'altra parte. In teoria faremmo 54*8=432 pixel
; di larghezza (monitor permettendo). Da notare che ci si riferisce a 8 pixel
; LOWRES, in caso di HIRES la dimensione rimane invariata, ed apparira' di
; 16 pixel hires, naturalmente.

;             /////////
;            /       /______________________
;           / ___ __//                      \
;          / ______//  eY! fig sta rutin!    \
;        _/ /  Ⓡ \ⓒ\\_ ______________________/
;       (_  \____/_/ / /
;        \ _       \ \/
;         \/    (·__)
;         /        |
;        /_____ (o)|
;          T T'----'
;          l_! xCz

LineCop:
        lea     TabellaColori(PC),a0
        lea     FineTabColori(PC),a3
        lea     EffInCop,a1     ; Indirizzo barra orizzontale 1
```

```
        lea     EffInCop2,a2    ; Indirizzo barra orizzontale 2
        moveq   #54-1,d3        ; Numero di colori orizzontali
        addq.l  #2,ColBarraAltOffset    ; Barra bassa - scorr. colori
                                        ; verso sinistra
        subq.l  #2,ColBarraBassOffset   ; Barra alta - scorrimento colori
                                        ; verso destra
        move.l  ColBarraAltOffset(PC),d0        ; Start Offset (1)
        add.l   d0,a0           ; trova il colore giusto nella tabella colori
                                ; secondo l'offset attuale
        cmp.w   #-1,(a0)        ; siamo alla fine della tabella? (indicata
                                ; con un dc.w -1)
        bne.s   CSalta          ; se no, vai avanti
        clr.l   ColBarraAltOffset       ; altrimenti riparti
        lea     TabellaColori(PC),a0    ; dal primo colore
CSalta:
        move.l  ColBarraBassOffset(PC),d1       ; Start Offset (2)
        sub.l   d1,a3                           ; trova il colore giusto
        cmp.w   #-1,-(a3)               ; siamo alla fine della tabella
        bne.s   MettiColori             ; se non ancora vai avanti
        move.l  #FineTabColori-TabellaColori,ColBarraBassOffset ; altrimenti
                                        ; fai ripartire dalla fine della
                                        ; tabella (dato che questa barra
                                        ; scorre all'indietro!)
        lea     FineTabColori-2(PC),a3
MettiColori:
        addq.w  #2,a1           ; salta il dc.w $180
        addq.w  #2,a2           ; salta il dc.w $180
        move.w  (a0)+,(a1)+     ; Immetti il colore in coplist (barra1)
        move.w  (a3),(a2)+      ; Immetti il col. nella barra 2

        cmp.w   #-1,(a0)        ; siamo alla fine della tabella colori? (bar1)
        bne.s   NonFine         ; se non ancora vai avanti
        lea     TabellaColori(PC),a0    ; altrimenti riparti da capo (bar1)
NonFine:
        cmp.w   #-1,-(a3)       ; siamo all'inizio della tab colori? (bar2)
        bne.s   NonFine2        ; se non ancora vai avanti
        lea     FineTabColori-2(PC),a3  ; altrimenti riparti dalla fine (bar2)
NonFine2:
        dbra    d3,MettiColori
        rts

*** *** *** *** *** *** *** *** *** ***


ColBarraAltOffset:
        dc.l    0

ColBarraBassOffset:
        dc.l    0



; NOTA: per indicare la fine ( e l'inizio) della tabella, viene controllato
;       se si e' arrivati al dc.w -1.

        dc.w    -1      ; fine tabella
TabellaColori:
        DC.W    $F0F,$F0E,$F0D,$F0C,$F0B,$F0A,$F09,$F08,$F07,$F06
        DC.W    $F05,$F04,$F03,$F02,$F01,$F00,$F10,$F20,$F30,$F40
        DC.W    $F50,$F60,$F70,$F80,$F90,$FA0,$FB0,$FC0,$FD0,$FE0
        DC.W    $FF0,$EF0,$DF0,$CF0,$BF0,$AF0,$9F0,$8F0,$7F0,$6F0
        DC.W    $5F0,$4F0,$3F0,$2F0,$1F0,$0F0,$0F1,$0F2,$0F3,$0F4
```

```
        DC.W    $0F5,$0F6,$0F7,$0F8,$0F9,$0FA,$0FB,$0FC,$0FD,$0FE
        DC.W    $0FF,$0EF,$0DF,$0CF,$0BF,$0AF,$09F,$08F,$07F,$06F
        DC.W    $05F,$04F,$03F,$02F,$01F,$00F,$10F,$20F,$30F,$40F
        DC.W    $50F,$60F,$70F,$80F,$90F,$A0F,$B0F,$C0F,$D0F,$E0F
FineTabColori:
        dc.w    -1      ; fine tabella


        section CList,code_c

CopperList:
        dc.w    $100,$200       ; BPLCON0 - 0 bitplanes
        dc.w    $180,$000       ; Color0 nero

        dc.w    $2901,$FFFE     ; Wait linea $29
EffInCop2:
        dcb.l   54,$1800000     ; 54 Color0 di seguito, che a scatti di 8
                                ; pixel in avanti ogni volta riempiono la
                                ; linea interamente

        dc.w    $2a01,$FFFE     ; Wait linea $2a
        dc.w    $180,$000       ; Color0 nero


        dc.w    $FFDF,$FFFE     ; Wait speciale per andare in zona PAL

        dc.w    $2A01,$FFFE     ; Attendi la linea $2a+$ff
EffInCop:
        dcb.l   54,$1800000     ; 54 Color0 di seguito, che a scatti di 8
                                ; pixel in avanti ogni volta riempiono la
                                ; linea interamente

        dc.w    $2B07,$FFFE     ; Wait linea $ff+$2b
        dc.w    $180,$000       ; Color0 nero

        dc.w    $FFFF,$FFFE     ; Fine copperlist

        end
```

Lezione11g5

```
; Lezione11g5.s -  Uso della caratteristica del copper di richiedere 8 pixel
;                  orizzontali per eseguire un suo "MOVE".
;                  Tasto destro per far scendere la "corda".

        SECTION Spago,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        bsr.w   FaiCopper       ; Crea la copperlist...
```

```
        lea     $dff000,a6
        MOVE.W  #DMASET,$96(a6)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPLIST,$80(a6)        ; Puntiamo la nostra COP
        move.w  d0,$88(a6)              ; Facciamo partire la COP
        move.w  #0,$1fc(a6)             ; Disattiva l'AGA
        move.w  #$c00,$106(a6)          ; Disattiva l'AGA
        move.w  #$11,$10c(a6)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        btst    #2,$16(a6)      ; tasto destro del mouse premuto?
        bne.s   NonScendere
        addq.b  #1,WaitLine     ; Se si, fai scendere il tutto!
NonScendere:

        bsr.w   MuoviCopper     ; rolla lo spago...

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts                     ; esci


*****************************************************************************
;       Routine che crea la copperlist. Per fare una linea orizzontale
;       completa occorrono 52 MOVE del copper. In questo caso prendiamo
;       alternativamente 32 MOVE per colore, per cui non terminano una
;       linea esatta, ma i 2 colori si "incrociano" in punti orizzontali
;       diversi. Questo crea una senzazione di "intreccio".
*****************************************************************************

NumeroIntrecci  EQU     8
IngombroCopEf   EQU     NumeroIntrecci*(32*2)


FaiCopper:
        LEA     CopBuf,A0               ; Indirizzo buffer in CopList
        MOVEQ   #NumeroIntrecci-1,D6    ; numero intrecci
MAIN0:
        LEA     COLORS1(PC),A1  ; tab COLORS1
        MOVEQ   #32-1,D7        ; 32 color0 per colori da COLORS1
COP0:
        MOVE.W  #$0180,(A0)+    ; registro COLOR0
        MOVE.W  (A1)+,(A0)+     ; valore del color dalla tabella COLORS1
        DBRA    d7,COP0         ; fai tutta la "linea" (non 1 intera...)
```

```
        LEA     COLORS2(PC),A1  ; tab COLORS2
        MOVEQ   #32-1,D7        ; 32 color0 per colori da COLORS2
COP1:
        MOVE.W  #$0180,(A0)+    ; registro COLOR0
        MOVE.W  (A1)+,(A0)+     ; valore del color0 dalla tabella COLORS2
        DBRA    d7,COP1         ; fai tutta la "linea" (non 1 intera...)
        DBRA    d6,MAIN0        ; Fai tutti gli "intrecci".
        RTS


COLORS1:
 DC.W   $003,$001,$002,$003,$004,$005,$006,$007
 DC.W   $008,$009,$00A,$00B,$00C,$00D,$00E,$10F
 DC.W   $10F,$00E,$00D,$00C,$00B,$00A,$009,$008
 DC.W   $007,$006,$005,$004,$003,$002,$001,$003

COLORS2:
 DC.W   $010,$010,$020,$030,$040,$050,$060,$070
 DC.W   $080,$090,$0A0,$0B0,$0C0,$0D0,$0E0,$0F0
 DC.W   $0F0,$0E0,$0D0,$0C0,$0B0,$0A0,$090,$080
 DC.W   $070,$060,$050,$040,$030,$020,$010,$010


*****************************************************************************
; Routine che "rotea" i colori...
*****************************************************************************

;          _
;        _( )_
;       (_-O-_)
;         (_)

MuoviCopper:
        LEA     CopBuf,A0       ; Buffer in copperlist
        MOVE.w  #IngombroCopEf-1,D7
        move.w  #(IngombroCopEf*4)-2,d6 ; offset per trovare l'ultimo colore
        MOVE.W  0(A0,D6.W),D0   ; ultimo colore in d0 (a0+offset!)
        MOVE.W  D6,D5
        SUBQ.W  #4,D5           ; offset colore precedente in d5
SYNC0:
        MOVE.W  0(A0,D5.W),0(A0,D6.W)   ; colore precedente in quello "dopo"
        SUBQ.W  #4,D6                   ; calcola offset prossimo colore
        SUBQ.W  #4,D5                   ; calcola offset prossimo colore
        dbra    d7,SYNC0                ; Esegui per tutto il "nodo"
        MOVE.W  D0,2(A0) ; metti l'ultimo colore, che avevamo salvato, come
                         ; primo colore, per non interrompere il ciclo.
        RTS

*****************************************************************************

        section coop,data_C

COPLIST:
        DC.W    $100,$200       ; BplCon0 - no bitplanes
        DC.W    $180,$003       ; Color0 - blu scuro
WaitLine:
        DC.W    $4001,$FFFE     ; Wait linea $40.
CopBuf:
        DCB.L   IngombroCopEf,0 ; Spazio per l'effetto cop
        DC.W    $180,3          ; Color0 - blu scuro
        DC.w    $ffff,$fffe     : Fine della Copperlist
```

```
        END
```

Un'altro utilizzo della peculiarita' dei move del copper per cui ognuno fa "scattare" in avanti di 8 pixel. Si puo' notare che tutto l'effetto e' composto solamente da decine di COLOR0 messi di seguito, per cui basta cambiare il wait che li precede per far spostare "tutto" in basso.

Lezione11g6

```
; Lezione11g6.s -  Uso della caratteristica del copper di richiedere 8 pixel
;                  orizzontali per eseguire un suo "MOVE".

        SECTION copfantasia,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

NUMLINES        =       80


START:
        MOVE.L  #$5001FFFE,D2   ; $50 = prima linea verticale
        BSR.W   MAKE_IT         ; fai la copper!

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPLIST,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0e000,d2      ; linea da aspettare = $e0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $e0
        BNE.S   Waity1

        BTST    #2,$16(a5)      ; tasto destro premuto?
        BEQ.s   Blocca

        BSR.w   FantaCop        ; rulla i colori...

Blocca:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
```

```
        rts

*****************************************************************************
*               Routine che crea la copperlist per l'effetto                *
*****************************************************************************

MAKE_IT:
        LEA     COPBUF,A0               ; Indirizzo copper buffer
        MOVEQ   #NUMLINES-1,D6          ; numero linee...
MAIN0:
        LEA     COLORS(PC),A1           ; Tabella con i colori...
        MOVEQ   #32-1,D7                ; Numero
        MOVE.L  D2,(A0)+                ; Metti il WAIT
        MOVE.L  #$01800505,(A0)+        ; color0
COP0:
        MOVE.W  #$0180,(A0)+    ; registro COLOR0
        MOVE.W  (A1)+,(A0)+     ; valore del color0 preso dalla tabella
        DBRA    d7,COP0          ; Fai una linea con 32 color0...
        MOVE.L  #$01800505,(A0)+ ; Metti un COLOR0
        ADDI.L  #$01020000,D2   ; Fai waitare 1 linea sotto e 2 piu' avanti
                                ; per creare la "diagonale".
        DBRA    d6,MAIN0        ; Fai tutte le linee
        RTS

; Tabella colori

COLORS:
 DC.W   $100,$101,$202,$303,$404,$505,$606,$707
 DC.W   $808,$909,$A0A,$B0B,$C0C,$D0D,$E0E,$F0F
 DC.W   $F0F,$E0E,$D0D,$C0C,$B0B,$A0A,$909,$808
 DC.W   $707,$606,$505,$404,$303,$202,$101,$100

*****************************************************************************
*               Routine che cicla i colori dell'effetto                     *
*****************************************************************************

FantaCop:
        LEA     COPBUF+8,A0     ; Indirizzo primo col da ciclare
        MOVEQ   #NUMLINES-1,D6  ; numero di linee da fare
MOVE1:
        MOVE.W  2(A0),D0        ; Salva il primo colore in d0
MOVE0:
        MOVE.W  2(A0),-2(A0)            ; copia i 32 colori della linea
        MOVE.W  6(A0),2(A0)             ; "indietro" di un posto.
        MOVE.W  6+4(A0),2+4(A0)
        MOVE.W  6+4*2(A0),2+4*2(A0)
        MOVE.W  6+4*3(A0),2+4*3(A0)
        MOVE.W  6+4*4(A0),2+4*4(A0)
        MOVE.W  6+4*5(A0),2+4*5(A0)
        MOVE.W  6+4*6(A0),2+4*6(A0)
        MOVE.W  6+4*7(A0),2+4*7(A0)
        MOVE.W  6+4*8(A0),2+4*8(A0)
        MOVE.W  6+4*9(A0),2+4*9(A0)
        MOVE.W  6+4*10(A0),2+4*10(A0)
        MOVE.W  6+4*11(A0),2+4*11(A0)
        MOVE.W  6+4*12(A0),2+4*12(A0)
        MOVE.W  6+4*13(A0),2+4*13(A0)
        MOVE.W  6+4*14(A0),2+4*14(A0)
        MOVE.W  6+4*15(A0),2+4*15(A0)
        MOVE.W  6+4*16(A0),2+4*16(A0)
        MOVE.W  6+4*17(A0),2+4*17(A0)
        MOVE.W  6+4*18(A0),2+4*18(A0)
```

```
        MOVE.W  6+4*19(A0),2+4*19(A0)
        MOVE.W  6+4*20(A0),2+4*20(A0)
        MOVE.W  6+4*21(A0),2+4*21(A0)
        MOVE.W  6+4*22(A0),2+4*22(A0)
        MOVE.W  6+4*23(A0),2+4*23(A0)
        MOVE.W  6+4*24(A0),2+4*24(A0)
        MOVE.W  6+4*25(A0),2+4*25(A0)
        MOVE.W  6+4*26(A0),2+4*26(A0)
        MOVE.W  6+4*27(A0),2+4*27(A0)
        MOVE.W  6+4*28(A0),2+4*28(A0)
        MOVE.W  6+4*29(A0),2+4*29(A0)
        MOVE.W  6+4*30(A0),2+4*30(A0)
        MOVE.W  6+4*31(A0),2+4*31(A0)
        lea     4*32(a0),A0     ; puntiamo alla prossima linea
        MOVE.W  D0,-(A0)        ; metti il primo colore salvato come ultimo
                                ; per non interrompere il ciclo.
        lea     14(a0),A0       ; saltiamo il wait+move "esterno"
        DBRA    d6,MOVE1        ; eseguiamo tutte le linee
        RTS

*****************************************************************************

        SECTION COPPY,DATA_C

COPLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes.
COPBUF:
        ds.b    NUMLINES*12+numlines*$20*4 ; spazio per l'effetto.
        dc.w    $ffff,$fffe             ; fine copperlist

        end
```

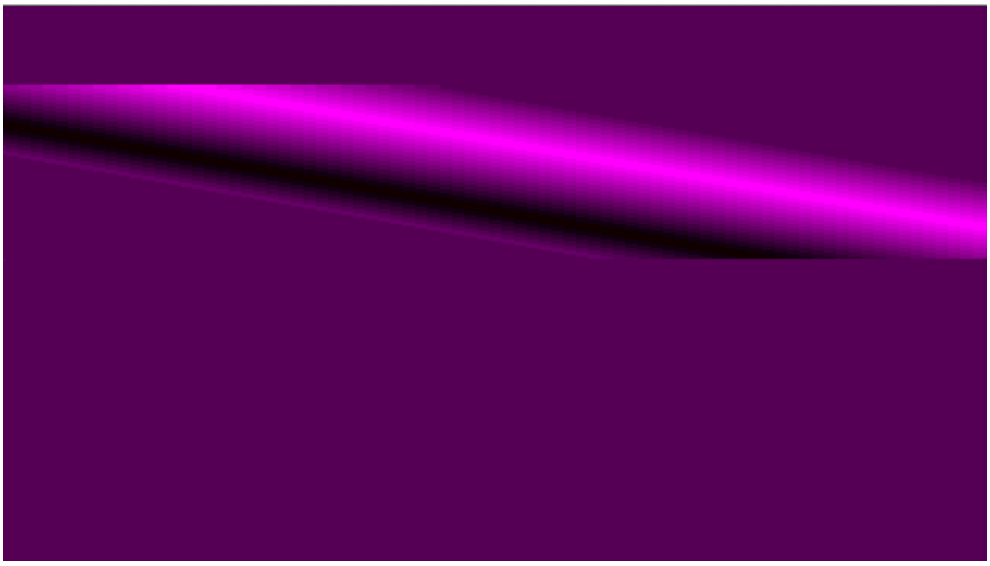


Figura 27.4: Lezione 11g6

Lezione11g7

```
; Lezione11g7.s -  Uso della caratteristica del copper di richiedere 8 pixel
;                  orizzontali per eseguire un suo "MOVE" per fare una specie
;                  di plasma. Tasto destro per bloccarlo.

        SECTION Plasmino,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        BSR.w   MAKEPLASM       ; Metti i colori dello "pseudoplasma"
        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0e000,d2      ; linea da aspettare = $e0
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $e0
        BNE.S   Waity1

        btst    #2,$dff016      ; tasto destro premuto?
        beq.s   Mouse2          ; se si non eseguire PLASMA

        BSR.w   PLASMA          ; Effetto PLASMA (piu' o meno...)

mouse2:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0e000,d2      ; linea da aspettare = $e0
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $e0
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************
;                       CREA I COLORI DEL PLASMA
*****************************************************************************
```

```
MAKEPLASM:
        LEA     PLASMZONE,A0    ; Indirizzo colori in coplist
        lea     COLCOPDAT,a1    ; tabella colori
        MOVEQ   #80-1,D7        ; numero linee
        moveq   #9,d2           ; e metti il contatore a 9
FaiLeLinee:
        MOVEQ   #52-1,D6        ; numero di COLOR0 per ogni linea
        ADDQ.w  #4,A0           ; Salta il WAIT tra una linea e l'altra
FaiUnaLinea:
        ADDQ.w  #2,A0           ; Salta il dc.w $180
        MOVE.W  (a1)+,(A0)+     ; per andare a piazzare il colore subito dopo
        DBRA    D6,FaiUnaLinea
        subq.b  #1,d2           ; segna che abbiamo fatto una linea
        bne.s   NonRipartire    ; se ne abbiamo fatte 8, d6=0, allora occorre
                                ; ripartire dal primo colore nella tabella.
        lea     COLCOPDAT(pc),a1 ; tab colori in a1 - riparti col colori.
        moveq   #9,d2           ; e metti il contatore a 9
NonRipartire:
        DBRA    D7,FaiLeLinee
        RTS


;       Tabella con i 52*9 colori di una linea orizzontale.

COLCOPDAT:
        dc.w    $26F,$27E,$28D,$29C,$2AB,$2BA,$2C9,$2D8,$2E7,$2F6
        dc.w    $4E7,$6D8,$8C9,$ABA,$CAA,$D9A,$E8A,$F7A,$F6C,$F5C
        dc.w    $D6D,$B6E,$96F,$76F,$56F,$36F,$26F,$27E,$28D,$29C
        dc.w    $2AB,$2BA,$2C9,$2D8,$2E7,$2F6,$4E7,$6D8,$8C9,$ABA
        dc.w    $CAA,$D9A,$E8A,$F7A,$F6B,$F5C,$D6D,$B6E,$96F,$76F
        dc.w    $56F,$36F,$36F,$37E,$38D,$39C,$3AB,$3BA,$3C9,$3D8
        dc.w    $3E7,$3F6,$4E7,$7D8,$9C9,$BBA,$DAA,$E9A,$F8A,$F7A
        dc.w    $F6C,$F5C,$E6D,$C6E,$A6F,$86F,$66F,$46F,$36F,$37E
        dc.w    $38D,$39C,$3AB,$3BA,$3C9,$3D8,$3E7,$3F6,$5E7,$7D8
        dc.w    $9C9,$BBA,$DAA,$E9A,$F8A,$F7A,$F6B,$F5C,$E6D,$C6E
        dc.w    $A6F,$86F,$46F,$46F,$36E,$37D,$38C,$39B,$3AA,$3B9
        dc.w    $3C8,$3D7,$3E6,$3F5,$4E6,$7D7,$9C8,$BB9,$DA9,$E99
        dc.w    $F89,$F79,$F6B,$F5B,$E6C,$C6D,$A6E,$86E,$66E,$46E
        dc.w    $36E,$37D,$38C,$39B,$3AA,$3B9,$3C8,$3D7,$3E6,$3F5
        dc.w    $5E6,$7D7,$9C8,$BB9,$DA9,$E99,$F89,$F79,$F6A,$F5B
        dc.w    $E6C,$C6E,$A6E,$86E,$46E,$46E,$46E,$47D,$48C,$49B
        dc.w    $4AA,$4B9,$4C8,$4D7,$4E6,$4F5,$5E6,$8D7,$AC8,$CB9
        dc.w    $EA9,$F99,$F89,$F79,$F6B,$F5B,$F6C,$D6D,$B6E,$96E
        dc.w    $76E,$56E,$46E,$47D,$48C,$49B,$4AA,$4B9,$4C8,$4D7
        dc.w    $4E6,$4F5,$6E6,$8D7,$AC8,$CB9,$EA9,$F99,$F89,$F79
        dc.w    $F6A,$F5B,$F6C,$D6E,$B6E,$96E,$56E,$56E,$45E,$46D
        dc.w    $47C,$48B,$49A,$4A9,$4B8,$4C7,$4D6,$4E5,$5D6,$8C7
        dc.w    $AB8,$CA9,$E99,$F89,$F79,$F69,$F5B,$F4B,$F5C,$D5D
        dc.w    $B5E,$95E,$75E,$55E,$45E,$46D,$47C,$48B,$49A,$4A9
        dc.w    $4B8,$4C7,$4D6,$4E5,$6D6,$8C7,$AB8,$CA9,$E99,$F89
        dc.w    $F79,$F69,$F5A,$F4B,$F5C,$D5E,$B5E,$95E,$55E,$55E
        dc.w    $44D,$45C,$46B,$47A,$489,$498,$4A7,$4B6,$4C5,$4D4
        dc.w    $5C5,$8B6,$AA7,$C98,$E88,$F78,$F68,$F68,$F59,$F4A
        dc.w    $F4B,$D4C,$B4D,$94D,$74D,$54D,$44D,$45C,$46B,$47A
        dc.w    $489,$498,$4A7,$4B6,$4C5,$4D4,$6C5,$8B6,$AA7,$C98
        dc.w    $E88,$F78,$F68,$F58,$F49,$F3A,$F4B,$D4D,$B4D,$94D
        dc.w    $54D,$54D,$44C,$45B,$46A,$479,$488,$499,$4A6,$4B5
        dc.w    $4C4,$4D3,$5C4,$8B5,$AA6,$C97,$E87,$F77,$F67,$F67
        dc.w    $F58,$F49,$F4C,$D4B,$B4C,$94C,$74C,$54C,$44C,$45B
        dc.w    $46A,$479,$488,$497,$4A6,$4B5,$4C4,$4D3,$6C4,$8B5
        dc.w    $AA6,$C97,$E87,$F77,$F67,$F57,$F48,$F39,$F4A,$D4C
```

```
        dc.w    $B4C,$94C,$54C,$54C,$44B,$45A,$469,$478,$487,$498
        dc.w    $4A5,$4B4,$4C3,$4D2,$5C3,$8B4,$AA5,$C96,$E86,$F76
        dc.w    $F66,$F66,$F57,$F48,$F4B,$D4A,$B4B,$94B,$74B,$54B
        dc.w    $44B,$45A,$469,$478,$487,$496,$4A5,$4B4,$4C3,$4D2
        dc.w    $6C3,$8B4,$AA5,$C96,$E86,$F76,$F66,$F56,$F47,$F38
        dc.w    $F49,$D4B,$B4B,$94B,$54B,$54B,$44A,$459,$468,$477
        dc.w    $486,$497,$4A4,$4B3,$4C2,$4D1,$5C2,$8B3,$AA4,$C95
        dc.w    $E85,$F75,$F65,$F65,$F56,$F47,$F4A,$D49,$B4A,$94A
        dc.w    $74A,$54A,$44A,$459,$468,$477,$486,$495,$4A4,$4B3
        dc.w    $4C2,$4D1,$6C2,$8B3,$AA4,$C95,$E85,$F75,$F65,$F55
        dc.w    $F46,$F37,$F48,$D4A,$B4A,$94A,$54A,$54A


PVAR1:
        dc.b    0
PVAR2:
        dc.b    0
PVAR3:
        dc.W    0

*****************************************************************************
; Questa routine, aziche' cambiare tutti i 54*80=4320 color0 della copperlist,
; cambia gli 80 WAIT tra una linea e l'altra, in modo da far iniziare le linee
; a posizioni orizzontali diverse. Questo sistema e' veloce, ma puo' far
; solamente scorrere a destra o a sinistra le linee. Un vero plasma lo vedremo
; piu' avanti.
*****************************************************************************

;          _________           :
;       ___\_      /           :
;       \_________/        __.____
;         |     |    .-  (_______)
;         !     _    :  __.,,,,,,`_
;        _-----(_)__ `- \_   _  __)
;       ( __  _  ___)     \__/  \_
;        \_/ /__/           \  __/
;         /    (_            )).
;         \_____/--------------'g®m


PLASMA:
        MOVEQ   #0,D1           ; azzeriamo i registri
        MOVEQ   #0,D3
        MOVEQ   #0,D5
        MOVE.B  PVAR1(PC),D1
        MOVE.B  PVAR2(PC),D4
        MOVE.B  PVAR3(PC),D5
        LEA     PLASMZONE,A0            ; Indirizzo plasma in copperlist
        LEA     PLASMMOVES(PC),A1       ; indirizzo tabella coi movimenti
        MOVE.W  #$8101,D3               ; Wait minimo da aggiungere
        MOVE.W  #79-1,D7                ; numero linee da "plasmizzare"
PLASMA2:
        MOVEQ   #0,D0                   ; azzeriamo d0
        MOVE.B  0(A1,D1.w),D0   ; e preleviamo dalla tabella in un modo un
        ADD.B   0(A1,D4.w),D0   ; po' incasinato il valore giusto secondo
        ADD.B   0(A1,D5.w),D0   ; le 3 variabili
        BCLR.L  #0,D0           ; non ci serve il bit basso
        ADD.W   D3,D0           ; aggiungiamo il wait MINIMO
        ADD.W   #$0100,D3       ; e spostiamo il wait minimo una linea sotto
        MOVE.W  D0,(A0)         ; Metti il WAIT cambiato
        ADD.W   #(52*4)+4,A0    ; e salta al prossimo WAIT
        ADD.B   #$FF,D1         ; seleziona solo il byte basso di d1
```

```
        ADDQ.B  #4,D4           ; aggiungi 4 a d4
        ADD.B   #$7D,D5         ; e 125 a d5 (che casino...)
        DBRA    D7,PLASMA2

        ADDQ.B  #1,PVAR1        ; secondo queste 3 variabili cambia il casino
        ADDQ.B  #2,PVAR2
        ADDQ.B  #1,PVAR3
        RTS


; parametri per il comando "IS":
;
; BEG> 0
; END> 180
; AMOUNT> 256
; AMPLITUDE> $4A
; YOFFSET> 0
; SIZE> B
; MULTIPLIER> 1

PLASMMOVES:     ; 256 bytes
 DC.B   $00,$01,$02,$03,$04,$05,$06,$07,$08,$09,$0A,$0A,$0B,$0C,$0D,$0E
 DC.B   $0F,$10,$11,$12,$12,$13,$14,$15,$16,$17,$18,$19,$19,$1A,$1B,$1C
 DC.B   $1D,$1E,$1E,$1F,$20,$21,$22,$22,$23,$24,$25,$26,$26,$27,$28,$29
 DC.B   $29,$2A,$2B,$2C,$2C,$2D,$2E,$2F,$2F,$30,$31,$31,$32,$33,$33,$34
 DC.B   $35,$35,$36,$37,$37,$38,$38,$39,$39,$3A,$3B,$3B,$3C,$3C,$3D,$3D
 DC.B   $3E,$3E,$3F,$3F,$40,$40,$41,$41,$41,$42,$42,$43,$43,$43,$44,$44
 DC.B   $45,$45,$45,$46,$46,$46,$46,$47,$47,$47,$47,$48,$48,$48,$48,$48
 DC.B   $49,$49,$49,$49,$49,$49,$49,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A
 DC.B   $4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$4A,$49,$49,$49,$49,$49,$49,$49
 DC.B   $48,$48,$48,$48,$48,$47,$47,$47,$47,$46,$46,$46,$46,$45,$45,$45
 DC.B   $44,$44,$43,$43,$43,$42,$42,$41,$41,$41,$40,$40,$3F,$3F,$3E,$3E
 DC.B   $3D,$3D,$3C,$3C,$3B,$3B,$3A,$39,$39,$38,$38,$37,$37,$36,$35,$35
 DC.B   $34,$33,$33,$32,$31,$31,$30,$2F,$2F,$2E,$2D,$2C,$2C,$2B,$2A,$29
 DC.B   $29,$28,$27,$26,$26,$25,$24,$23,$22,$22,$21,$20,$1F,$1E,$1E,$1D
 DC.B   $1C,$1B,$1A,$19,$19,$18,$17,$16,$15,$14,$13,$12,$12,$11,$10,$0F
 DC.B   $0E,$0D,$0C,$0B,$0A,$0A,$09,$08,$07,$06,$05,$04,$03,$02,$01,$00

*****************************************************************************

        Section Copper,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$000       ; color0 nero
        dc.w    $7F19,$FFFE
        dc.w    $180,$ff0       ; giallo (cornicina)
        dc.w    $8007,$FFFE
        dc.w    $180,$06a       ; colore intermedio
PLASMZONE:
                                ; Questo pezzo di copperlist lo avremmo potuto
                                ; fare anche con una routine...
        dc.l    $8101FFFE       ; wait
        dcb.l   52,$1800000     ; 54*color0
        dc.l    $8201FFFE       ; wait
        dcb.l   52,$1800000     ; 54*color0
        dc.l    $8301FFFE       ; eccetera
        dcb.l   52,$1800000
        dc.l    $8401FFFE
        dcb.l   52,$1800000
        dc.l    $8501FFFE
        dcb.l   52,$1800000
```

```
        dc.l    $8601FFFE
        dcb.l   52,$1800000
        dc.l    $8701FFFE
        dcb.l   52,$1800000
        dc.l    $8801FFFE
        dcb.l   52,$1800000
        dc.l    $8901FFFE
        dcb.l   52,$1800000
        dc.l    $8A01FFFE
        dcb.l   52,$1800000
        dc.l    $8B01FFFE
        dcb.l   52,$1800000
        dc.l    $8C01FFFE
        dcb.l   52,$1800000
        dc.l    $8D01FFFE
        dcb.l   52,$1800000
        dc.l    $8E01FFFE
        dcb.l   52,$1800000
        dc.l    $8F01FFFE
        dcb.l   52,$1800000
        dc.l    $9001FFFE
        dcb.l   52,$1800000
        dc.l    $9101FFFE
        dcb.l   52,$1800000
        dc.l    $9201FFFE
        dcb.l   52,$1800000
        dc.l    $9301FFFE
        dcb.l   52,$1800000
        dc.l    $9401FFFE
        dcb.l   52,$1800000
        dc.l    $9501FFFE
        dcb.l   52,$1800000
        dc.l    $9601FFFE
        dcb.l   52,$1800000
        dc.l    $9701FFFE
        dcb.l   52,$1800000
        dc.l    $9801FFFE
        dcb.l   52,$1800000
        dc.l    $9901FFFE
        dcb.l   52,$1800000
        dc.l    $9A01FFFE
        dcb.l   52,$1800000
        dc.l    $9B01FFFE
        dcb.l   52,$1800000
        dc.l    $9C01FFFE
        dcb.l   52,$1800000
        dc.l    $9D01FFFE
        dcb.l   52,$1800000
        dc.l    $9E01FFFE
        dcb.l   52,$1800000
        dc.l    $9F01FFFE
        dcb.l   52,$1800000
        dc.l    $A001FFFE
        dcb.l   52,$1800000
        dc.l    $A101FFFE
        dcb.l   52,$1800000
        dc.l    $A201FFFE
        dcb.l   52,$1800000
        dc.l    $A301FFFE
        dcb.l   52,$1800000
        dc.l    $A401FFFE
        dcb.l   52,$1800000
```

```
        dc.l    $A501FFFE
        dcb.l   52,$1800000
        dc.l    $A601FFFE
        dcb.l   52,$1800000
        dc.l    $A701FFFE
        dcb.l   52,$1800000
        dc.l    $A801FFFE
        dcb.l   52,$1800000
        dc.l    $A901FFFE
        dcb.l   52,$1800000
        dc.l    $AA01FFFE
        dcb.l   52,$1800000
        dc.l    $AB01FFFE
        dcb.l   52,$1800000
        dc.l    $AC01FFFE
        dcb.l   52,$1800000
        dc.l    $AD01FFFE
        dcb.l   52,$1800000
        dc.l    $AE01FFFE
        dcb.l   52,$1800000
        dc.l    $AF01FFFE
        dcb.l   52,$1800000
        dc.l    $B001FFFE
        dcb.l   52,$1800000
        dc.l    $B101FFFE
        dcb.l   52,$1800000
        dc.l    $B201FFFE
        dcb.l   52,$1800000
        dc.l    $B301FFFE
        dcb.l   52,$1800000
        dc.l    $B401FFFE
        dcb.l   52,$1800000
        dc.l    $B501FFFE
        dcb.l   52,$1800000
        dc.l    $B601FFFE
        dcb.l   52,$1800000
        dc.l    $B701FFFE
        dcb.l   52,$1800000
        dc.l    $B801FFFE
        dcb.l   52,$1800000
        dc.l    $B901FFFE
        dcb.l   52,$1800000
        dc.l    $BA01FFFE
        dcb.l   52,$1800000
        dc.l    $BB01FFFE
        dcb.l   52,$1800000
        dc.l    $BC01FFFE
        dcb.l   52,$1800000
        dc.l    $BD01FFFE
        dcb.l   52,$1800000
        dc.l    $BE01FFFE
        dcb.l   52,$1800000
        dc.l    $BF01FFFE
        dcb.l   52,$1800000
        dc.l    $C001FFFE
        dcb.l   52,$1800000
        dc.l    $C101FFFE
        dcb.l   52,$1800000
        dc.l    $C201FFFE
        dcb.l   52,$1800000
        dc.l    $C301FFFE
        dcb.l   52,$1800000
```

```
        dc.l    $C401FFFE
        dcb.l   52,$1800000
        dc.l    $C501FFFE
        dcb.l   52,$1800000
        dc.l    $C601FFFE
        dcb.l   52,$1800000
        dc.l    $C701FFFE
        dcb.l   52,$1800000
        dc.l    $C801FFFE
        dcb.l   52,$1800000
        dc.l    $C901FFFE
        dcb.l   52,$1800000
        dc.l    $CA01FFFE
        dcb.l   52,$1800000
        dc.l    $CB01FFFE
        dcb.l   52,$1800000
        dc.l    $CC01FFFE
        dcb.l   52,$1800000
        dc.l    $CD01FFFE
        dcb.l   52,$1800000
        dc.l    $CE01FFFE
        dcb.l   52,$1800000
        dc.l    $CF01FFFE
        dcb.l   52,$1800000
        dc.l    $D001FFFE
        dcb.l   52,$1800000
        dc.l    $D101FFFE

        dc.l    $D219FFFE       ; Wait $d2
        dc.l    $1800FF0        ; Color0 giallo (cornicina)
        dc.l    $D311FFFE       ; Wait $d3
        dc.l    $1800000        ; Color0 nero
        dc.l    $FFFFFFFE       ; fine della copperlist

        END
```

27.8 Lezione11h

```
; Lezione11h.s  Uso del COP2LC ($dff084) per fare una copperlist dinamica,
;               ossia una copperlist che ogni frame si alternano 2 copperlist
;               fatte in modo che "aumenti" la credibilita' di una sfumatura.
;               Tasto destro per vedere la differenza!

        SECTION DynaCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
;        PUNTIAMO IL NOSTRO BITPLANE
```

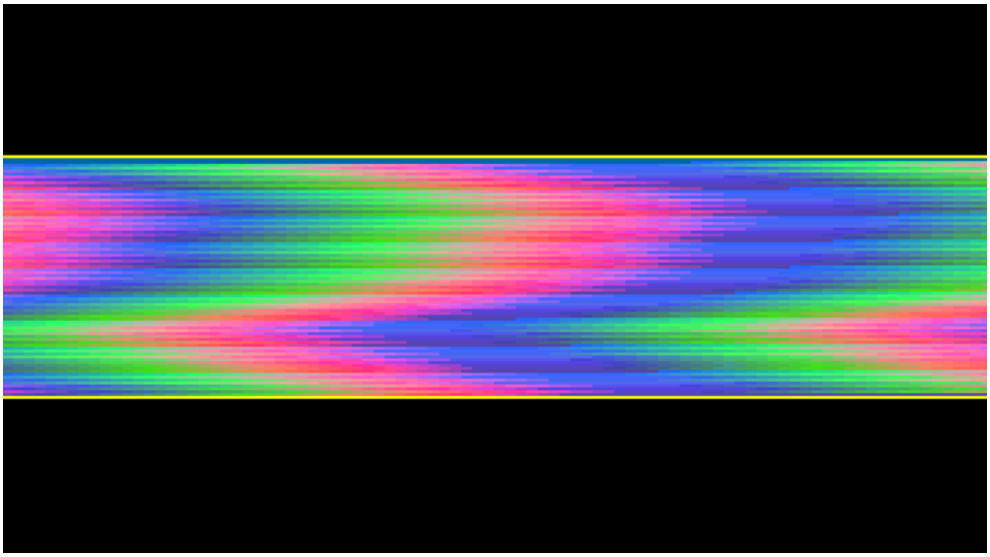


Figura 27.5: Lezione 11g7

```
        MOVE.L  #BITPLANE,d0
        LEA     BPLPOINTERS,A1
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
        dbra    d1,NulLoop

        bsr.w   InitCops        ; Crea le 2 copperlist da "scambiare"

        lea     $dff000,a6
        MOVE.W  #DMASET,$96(a6)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a6)     ; Puntiamo la nostra COP
        move.w  d0,$88(a6)              ; Facciamo partire la COP
        move.w  #0,$1fc(a6)             ; Disattiva l'AGA
        move.w  #$c00,$106(a6)          ; Disattiva l'AGA
        move.w  #$11,$10c(a6)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
```

```
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        btst    #2,$16(a6)      ; tasto destro del mouse premuto?
        beq.s   NonSwappare     ; Se si non scambiare (bello schifo!)

        movem.l CoppPointer1(PC),d0-d1  ; Metti con un solo MOVEM gli indirizzi
                                        ; delle 2 copperlist in d0 e in a1
        move.l  d0,CoppPointer2         ; Scambiane l'ordine...
        move.l  d1,CoppPointer1         ; ...
        move.w  d1,Cop2lcl              ; E punta l'altra copperlist2 come
        swap    d1                      ; prossima a cui saltare con
        move.w  d1,Cop2lch              ; il COPJMP2 ($dff08a)
nonSwappare:

        bsr.w   PrintCarattere  ; Stampa un carattere alla volta

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts                     ; esci


CoppPointer1:
                dc.l    ColInt1
CoppPointer2:
                dc.l    ColInt2

*****************************************************************************
* Routine che crea le 2 copperslist che dovranno essere visualizzate        *
* alternativamente puntandole nel COP2LC e facendole partire con COPJMP2    *
*****************************************************************************

;         _________________
;        /                 \
;        \   ___     ___   /
;       __\  (__)   (__)  /__
;       \__\___  ' '  ___/__/
;            \ \...../ /
;             \_______/g®m


COLSTART        EQU     $660    ; Colore di partenza = giallo
COLTENDENZA     EQU     $001    ; Tendenza (valore aggiunto ogni waitata)

InitCops:
        move.l  #$4407fffe,d0   ; Wait - inizia dalla linea orizzontale $44
        move.l  #$1800000,d1    ; Color0
        move.w  #COLSTART,d2    ; Colore di partenza
        move.w  #COLTENDENZA,d3 ; tendenza destinazione ($001/$010/$100)
        moveq   #2-1,d5         ; 2 Copperlist da fare
        lea     ColInt1,a1      ; Prima Copperlist
makecop:
```

```
        move.w  d2,d1           ; Copia colstart in d1 (nel $180xxxx!)
        move.l  d0,(a1)+        ; Metti il WAIT in coplist
        move.l  d1,(a1)+        ; Metti il $180xxxx (color0) in coplist
        add.l   #$05000000,d0   ; wait 5 linee piu' in basso la volta dopo
        move.l  d0,(a1)+        ; metti wait
        move.l  d1,(a1)+        ; metti il $180xxxx (color0)
        add.l   #$05000000,d0   ; wait 5 linee piu' in basso
        move.w  d2,d4           ; copia il colstart in d4
        and.w   #$00f,d4        ; Seleziona solo la componente BLU
        cmp.w   #$00f,d4        ; e' al massimo?
        beq.S   endcop          ; Se si, endcop!
        move.w  d2,d4           ; Altrimenti, vediamo il verde:
        and.w   #$0f0,d4        ; seleziona solo la componente verde.
        cmp.w   #$0f0,d4        ; E' al massimo?
        beq.S   endcop          ; Se si, endcop!
        move.w  d2,d4
        and.w   #$f00,d4        ; Seleziona solo la componente ROSSA
        cmp.w   #$f00,d4        ; e' al massimo?
        beq.S   endcop          ; Se si ENDCOP!
        add.w   d3,d2           ; Aggiungi COLTENDENZA al COLORSTART
        bra.S   makecop         ; E continua...
endcop:
        move.l  #$fffffffe,d0   ; Fine copperlist in d0
        move.w  d2,d1           ; copia COLORSTART in d1
        move.l  d0,(a1)+        ; fine copperlist
        move.l  #$4907fffe,d0
        move.l  #$1800000,d1
        move.w  #COLSTART,d2
        move.w  #COLTENDENZA,d3
        lea     ColInt2,a1
        dbf     d5,makecop
        rts


*****************************************************************************
;                       Routine di Print
*****************************************************************************

PRINTcarattere:
        movem.l d2/a0/a2-a3,-(SP)
        MOVE.L  PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        CMP.B   #$ff,d2         ; Segnale di fine testo? ($FF)
        beq.s   FineTesto       ; Se si, esci senza stampare
        TST.B   d2              ; Segnale di fine riga? ($00)
        bne.s   NonFineRiga     ; Se no, non andare a capo

        ADD.L   #40*7,PuntaBITPLANE     ; ANDIAMO A CAPO
        ADDQ.L  #1,PuntaTesto           ; primo carattere riga dopo
                                        ; (saltiamo lo ZERO)
        move.b  (a0)+,d2                ; primo carattere della riga dopo
                                        ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        LSL.W   #3,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D2,A2
```

```
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L  PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,40*3(A3)  ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,40*4(A3)  ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,40*5(A3)  ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,40*6(A3)  ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,40*7(A3)  ; stampa LA LINEA 8  " "

        ADDQ.L  #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L  #1,PuntaTesto   ; prossimo carattere da stampare

FineTesto:
        movem.l (SP)+,d2/a0/a2-a3
        RTS


PuntaTesto:
        dc.l    TESTO

PuntaBitplane:
        dc.l    BITPLANE

;       $00 per "fine linea" - $FF per "fine testo"

                ; numero caratteri per linea: 40
TESTO:       ;            1111111111222222222233333333334
             ;   12345678901234567890123456789012345678890
        dc.b    '                                        ',0 ; 1
        dc.b    '    Questo listato utilizza il COP2LC   ',0 ; 2
        dc.b    '                                        ',0 ; 3
        dc.b    '    ($dff084) per far saltare, ad una   ',0 ; 4
        dc.b    '                                        ',0 ; 5
        dc.b    '    certa linea video, ad un altra      ',0 ; 6
        dc.b    '                                        ',0 ; 7
        dc.b    '    copperlist. Al termine di questa    ',0 ; 8
        dc.b    '                                        ',0 ; 9
        dc.b    '    riparte sempre e comunque la        ',0 ; 10
        dc.b    '                                        ',0 ; 11
        dc.b    '    copperlist 1 (in $dff180). Dunque   ',0 ; 12
        dc.b    '                                        ',0 ; 13
        dc.b    '    basta cambiare solo la cop2 a cui   ',0 ; 14
        dc.b    '                                        ',0 ; 15
        dc.b    '    puntare ogni frame, per DynamiCop!  ',0 ; 16
        dc.b    '                                        ',0 ; 17
        dc.b    '    Il tasto destro ferma lo scambio.   ',$FF ; 18

        EVEN

;       Il FONT caratteri 8x8 (copiato in CHIP dalla CPU e non dal blitter,
;       per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin  "assembler2:sorgenti4/nice.fnt"

*****************************************************************************
```

```
        Section copperDynamic,data_C

copperlist:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
                    ; 5432109876543210
        dc.w    $100,%0001001000000000  ; 1 bitplane LOWRES 320x256

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ;primo   bitplane

        dc.w    $180,COLSTART   ; COLOR0 - colore di "partenza"
        dc.w    $182,$FF0       ; color1 - SCRITTE

;       dc.w    $2ce1,$fffe     ; Waitiamo almeno Y=$2c X=$d7

        dc.w    $84             ; registro COP2LCH (indirizzo copper 2!)
COP2LCH:
        dc.w    0
        dc.w    $86             ; registro COP2LCL
COP2LCL:
        dc.w    0

        dc.w    $8a,$000        ; COPJMP2 - fai partire la copperlist 2

*****************************************************************************

; spazio per la copperlist 1

ColInt1:
        dcb.l   2*60,0

*****************************************************************************

; spazio per la copperlist 2

ColInt2:
        dcb.l   2*60,0


*****************************************************************************

        SECTION MIOPLANE,BSS_C

BITPLANE:
        ds.b    40*256  ; un bitplane lowres 320x256

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
        ds.l    4               ; negli eventuali puntatori inutilizzati
```

```
        END
```

Figura 27.6: Lezione 11h

Lezione11h2

```
; Lezione11h2.s - Routine che genera delle barre sfumate - USARE IL TASTO
;                 DESTRO DEL MOUSE PER AUMENTARE l'ALTEZZA DELLE BARRE.

        SECTION Barrex,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

LINEE:  equ     211

START:
        bsr.s   FaiCopp1

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #OURCOPPER,$80(a5)      ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
```

```
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10500,d2      ; linea da aspettare = $105
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $105
        BNE.S   Waity1

        BSR.s   changecop       ; chiama la routine che cambia il copper

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************
; routine che crea la copperlist
*****************************************************************************

FaiCopp1:
        LEA     copcols,a0      ; indirizzo buffer in copperlist
        MOVE.L  #$2c07fffe,d1   ; istruzione copper wait, che inizia
                                ; attendendo alla linea $2c
        MOVE.L  #$1800000,d2    ; $dff180 = colore 0 per il copper
        MOVE.w  #LINEE-1,d0     ; numero di linee per il loop
        MOVEQ   #$000,d3        ; colore da mettere = nero
coploop:
        MOVE.L  d1,(a0)+        ; Metti il WAIT
        MOVE.L  d2,(a0)+        ; Metti il $180 (color0) azzerato al NERO
        ADD.L   #$01000000,d1   ; Fai aspettare il WAIT 1 linea dopo
        DBRA    d0,coploop      ; ripeti fino alla fine delle linee
        rts

*****************************************************************************
; routine che cambia i colori nella copperlist
*****************************************************************************

changecop:
        btst    #2,$dff016      ; tasto destro premuto?
        bne.s   noadd           ; se no, salta a noadd
        cmp.b   #$24,barlen     ; altrimenti controlla se siamo gia' a $24
        beq.s   noadd           ; in tal caso salta a noadd
        addq.b  #1,barlen       ; oppure ingrandisci la barra (BARLEN)
noadd:
        LEA     copcols,a0      ; indirizzo buffer in copperlist
        MOVE.w  #LINEE-1,d0     ; numero linee per il loop
        MOVE.L  PuntatoreTABCol(PC),a1  ; inizio della tabella colori in a1
        move.l  a1,PuntatTemporaneo     ; salvato nel PuntatoreTemporaneo
        moveq   #0,d1                   ; azzero d1
LineeLoop:
        move.w  (a1)+,6(a0)     ; copia il colore dalla tabella alla copperlist
        addq.w  #8,a0           ; prossimo color0 in copperlist
        addq.b  #1,d1           ; annoto in d1 la lunghezza della sotto-barra
        cmp.b   barlen(PC),d1   ; fine della sotto-barra?
        bne.s   AspettaSottoBarra

        MOVE.L  PuntatTemporaneo(PC),a1
        addq.w  #2,a1                   ; punto al colore dopo
        cmp.l   #FINETABColBarra,PuntatTemporaneo       ; siamo a fine tab?
```

```
        bne.s   NonRipartire            ; se non ancora, vai a NonRipartire
        lea     TABColoriBarra(pc),a1   ; altrimenti riparti dal primo col!
NonRipartire:
        move.l  a1,PuntatTemporaneo     ; e salva il valore nel Pun. temporaneo
        moveq   #0,d1                   ; azzero d1
AspettaSottoBarra:
        dbra d0,LineeLoop       ; fai tutte le linee


        addq.l  #2,PuntatoreTABCol              ; prossimo colore
        cmp.l   #FINETABColBarra+2,PuntatoreTABCol ; siamo alla fine della
                                                ; tabella colori?
        bne.s FineRoutine                       ; se no, esci, altrimenti...
        move.l #TABColoriBarra,PuntatoreTABCol  ; riparti dal primo valore di
                                                ; TABColoriBarra
FineRoutine:
        rts

;       altezza barre

barlen:
        dc.b    1

        even


;       Tabella con i valori RGB dei colori. in questo caso sono toni di BLU

TABColoriBarra:
        dc.w    $000,$001,$002,$003,$004,$005,$006,$007
        dc.w    $008,$009,$00A,$00B,$00C,$00D,$00D,$00E
        dc.w    $00E,$00F,$00F,$00F,$00E,$00E,$00D,$00D
        dc.w    $00C,$00B,$00A,$009,$008,$007,$006,$005
        dc.w    $004,$003,$002,$001,$000,$000,$000,$000
        dcb.w   10,$000
FINETABColBarra:
        dc.w    $000,$001,$002,$003,$004,$005,$006,$007 ; questi valori servono
        dc.w    $008,$009,$00A,$00B,$00C,$00D,$00D,$00E ; per le sotto-barre
        dc.w    $00E,$00F,$00F,$00F,$00E,$00E,$00D,$00D
        dc.w    $00C,$00B,$00A,$009,$008,$007,$006,$005
        dc.w    $004,$003,$002,$001,$000,$000,$000,$000


PuntatTemporaneo:
        dc.l    TABColoriBarra

PuntatoreTABCol:
        DC.L    TABColoriBarra

*****************************************************************************

        Section Coppy,data_C

OURCOPPER:
        dc.w    $180,$000       ; Color0 nero
        dc.w    $100,$200       ; bplcon0 - no bitplanes

copcols:
        dcb.b   LINEE*8,0       ; spazio per 100 linee in questo formato:
                                ; WAIT xx07,$fffe
                                ; MOVE $xxx,$180        ; color0
        dc.w    $ffdf,$fffe
```

```
        dc.w    $0107,$fffe
        dc.w    $180,$010
        dc.w    $0207,$fffe
        dc.w    $180,$020
        dc.w    $0307,$fffe
        dc.w    $180,$030
        dc.w    $0507,$fffe
        dc.w    $180,$040
        dc.w    $0707,$fffe
        dc.w    $180,$050
        dc.w    $0907,$fffe
        dc.w    $180,$060
        dc.w    $0c07,$fffe
        dc.w    $180,$070
        dc.w    $0f07,$fffe
        dc.w    $180,$080
        dc.w    $1207,$fffe
        dc.w    $180,$090
        dc.w    $1507,$fffe
        dc.w    $180,$0a0

        dc.w    $180,$000       ; color0 nero
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

        end
```

Lezione11h3

```
; Lezione11h3.s - Effetto copper PRECALCOLATO!!! 50 copperlist precalcolate
;                 e visualizzate in sequenza usando COP2LC ($dff084).

        SECTION BarrexPrecalc,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        bsr.w   precalcop       ; Routine che precalcola 50 copperlist per
                                ; fare un "loop" completo dell'effetto

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #OURCOPPER,$80(a5)      ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
```

```
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        btst    #2,$dff016      ; tasto destro premuto?
        beq.s   Mouse2          ; se si non eseguire SwappaCoppero

        bsr.w   SwappaCoppero   ; Fai puntare alla prossima copperlist per
                                ; la corretta "animazione" dell'effetto.

mouse2:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts


*****************************************************************************
; Routine di SWAP delle copperlist precalcolate.. (un'animazione!!!)
*****************************************************************************

;             \||||/
;             (·)(·)
;          ___\ \/ /___
;         (_   \__/   _)
;         /   .        \
;        /    |Stz!\_   \
;       (_____| ___(_____)
;        (___)      (___)
;          ¡   \_    ¡
;          |_   |   _|
;          |    ;    |
;          |    |    |
;        __(____|____)__
;       (_______:_______)


SwappaCoppero:
        MOVE.L  copbufpunt(PC),D0
        lea     coppajumpa,a0   ; indirizzo puntatori al cop2 in coplist
        MOVE.W  D0,6(A0)        ; fai puntare all'attuale fotogramma
        SWAP    D0
        MOVE.W  D0,2(A0)
        ADD.L   #(linee*8)+AGGIUNTE,COPBUFPUNT  ; punta alla PROSSIMA COP
        MOVE.L  copbufpunt(PC),D0
        cmp.l   #finebuffoni,d0         ; Siamo all'ultima copper?
        bne.w   NonRibuffona            ; Se non ancora, ok
        move.l  #copbuf1,copbufpunt     ; altrimenti riparti dalla prima!
NonRibuffona:
        rts

*****************************************************************************
```

```
;              routine di precalc dell'effetto copper
*****************************************************************************

;       /\____/\
;       \(O..O)/
;        (----)
;         TTTT Mo!

LINEE           equ     211
AGGIUNTE        equ     20      ; LUNGHEZZA PARTI AGGIUNTE IN FONDO...
NUMBUFCOPPERI   equ     50      ; numero fotogrammi/copperlist!!!

PrecalCop:

; Ora creiamo le copperlist.

        lea     copbuf1,a0              ; Indirizzo buffers dove fare cops
        move.w  #NUMBUFCOPPERI-1,d7     ; numero di copperlist da precalcolare
FaiBuf:
        bsr.w   FaiCopp1                ; Fai una copperlist
        add.w   #(linee*8)+AGGIUNTE,a0  ; punta a quella dopo
        dbra    d7,FaiBuf               ; fai tutti i fotogrammi

; Ora le "riempiamo", come se eseguissimo l'effetto in real-time.

        move.w  #NUMBUFCOPPERI-1,d7     ; numero cops da "riempire"
        lea     copbuf1,a0              ; indirizzo prima copper precalcolata
ribuf:
        BSR.s   changecop       ; chiama la routine che cambia il copper
        add.w   #(linee*8)+AGGIUNTE,a0 ; salta alla prossima da riempire
        dbra    d7,riBuf        ; riempi tutte le copperlists


; Infine puntiamo le copperlists nei puntatori in copperlist!!!

        MOVE.L  #copbuf1,D0     ; primo "fotogramma" copper2
        lea     coppajumpa,a0   ; puntatore alla fine di copper1
        MOVE.W  D0,6(A0)        ; puntiamo...
        SWAP    D0
        MOVE.W  D0,2(A0)

        MOVE.L  #ourcopper,D0   ; copper1 INIZIO
        lea     coppajumpa2,a0  ; puntatore alla fine della "pezzofinale"
        MOVE.W  D0,6(A0)        ; a cui salta la copper2.
        SWAP    D0
        MOVE.W  D0,2(A0)
        rts

*****************************************************************************
; routine che crea una copperlist
*****************************************************************************

FaiCopp1:
        move.l  a0,-(SP)
        MOVE.L  #$2c07fffe,d1   ; istruzione copper wait, che inizia
                                ; attendendo alla linea $2c
        MOVE.L  #$1800000,d2    ; $dff180 = colore 0 per il copper
        MOVE.w  #LINEE-1,d0     ; numero di linee per il loop
        MOVEQ   #$000,d3        ; colore da mettere = nero
coploop:
        MOVE.L  d1,(a0)+        ; Metti il WAIT
        MOVE.L  d2,(a0)+        ; Metti il $180 (color0) azzerato al NERO
```

```
        ADD.L   #$01000000,d1   ; Fai aspettare il WAIT 1 linea dopo
        DBRA    d0,coploop      ; ripeti fino alla fine delle linee
        move.l  finPunt(PC),d0  ; pezzofinale, a cui saltano tutte le copper2
                                ; usate come fotogrammi.
        MOVE.w  #$82,(A0)+      ; PARTEFINALE da puntare - COP1LC
        move.w  d0,(a0)+
        swap    d0
        MOVE.w  #$80,(A0)+
        move.w  d0,(a0)+
        move.l  #$880000,(a0)+  ; COPJMP1 - salto al pezzo finale, il quale
                                ; poi ristabilira' copper1 come prima cop!
        move.l  (SP)+,a0
        rts

CopBufPunt:
        dc.l    copbuf1
FinPunt:
        dc.l    pezzofinale

*****************************************************************************
; routine che cambia i colori in una copperlist
*****************************************************************************

changecop:
        move.l  a0,-(SP)        ; salva a0 nello stack
        MOVE.w  #LINEE-1,d0     ; numero linee per il loop
        MOVE.L  PuntatoreTABCol(PC),a1  ; inizio della tabella colori in a1
        move.l  a1,PuntatTemporaneo     ; salvato nel PuntatoreTemporaneo
        moveq   #0,d1                   ; azzero d1
LineeLoop:
        move.w  (a1)+,6(a0)     ; copia il colore dalla tabella alla copperlist
        addq.w  #8,a0           ; prossimo color0 in copperlist
        addq.b  #1,d1           ; annoto in d1 la lunghezza della sotto-barra
        cmp.b   #9,d1           ; fine della sotto-barra?
        bne.s   AspettaSottoBarra

        MOVE.L  PuntatTemporaneo(PC),a1
        addq.w  #2,a1                   ; punto al colore dopo
        cmp.l   #FINETABColBarra,PuntatTemporaneo       ; siamo a fine tab?
        bne.s   NonRipartire            ; se non ancora, vai a NonRipartire
        lea     TABColoriBarra(pc),a1   ; altrimenti riparti dal primo col!
NonRipartire:
        move.l  a1,PuntatTemporaneo     ; e salva il valore nel Pun. temporaneo
        moveq   #0,d1                   ; azzero d1
AspettaSottoBarra:
        dbra d0,LineeLoop       ; fai tutte le linee

        addq.l  #2,PuntatoreTABCol              ; prossimo colore
        cmp.l   #FINETABColBarra+2,PuntatoreTABCol ; siamo alla fine della
                                                ; tabella colori?
        bne.s FineRoutine                       ; se no, esci, altrimenti...
        move.l #TABColoriBarra,PuntatoreTABCol  ; riparti dal primo valore di
                                                ; TABColoriBarra
FineRoutine:
        move.l  (SP)+,a0        ; riprendi a0 dallo stack
        rts

;       Tabella con i valori RGB dei colori. in questo caso sono toni di BLU

TABColoriBarra:
        dc.w    $000,$001,$002,$003,$004,$005,$006,$007
        dc.w    $008,$009,$00A,$00B,$00C,$00D,$00D,$00E
```

```
        dc.w    $00E,$00F,$00F,$00F,$00E,$00E,$00D,$00D
        dc.w    $00C,$00B,$00A,$009,$008,$007,$006,$005
        dc.w    $004,$003,$002,$001,$000,$000,$000,$000
        dcb.w   10,$000
FINETABColBarra:
        dc.w    $000,$001,$002,$003,$004,$005,$006,$007 ; questi valori servono
        dc.w    $008,$009,$00A,$00B,$00C,$00D,$00D,$00E ; per le sotto-barre
        dc.w    $00E,$00F,$00F,$00F,$00E,$00E,$00D,$00D
        dc.w    $00C,$00B,$00A,$009,$008,$007,$006,$005
        dc.w    $004,$003,$002,$001,$000,$000,$000,$000


PuntatTemporaneo:
        dc.l    TABColoriBarra

PuntatoreTABCol:
        DC.L    TABColoriBarra

*****************************************************************************

        SECTION GRAPH,DATA_C

ourcopper:
Copper2:
        dc.w    $180,$000       ; Color0 - nero
        dc.w    $100,$200       ; BplCon0 - no bitplanes

; qua potete mettere spritepointers, colori, bplpointers eccetera...


coppajumpa:
        dc.w    $84             ; COP2LCh
        DC.W    0
        dc.w    $86             ; COP2LCl
        DC.W    0
        DC.W    $8a,0           ; COPJMP2 - fai partire la cop2 (fotogramma)

* * * * * *

pezzofinale:                    ; a questo pezzo salta la copper2 alla sua
        dc.w    $ffdf,$fffe     ; fine, ogni fotogramma di animazione...
        dc.w    $0107,$fffe
        dc.w    $180,$010
        dc.w    $0207,$fffe
        dc.w    $180,$020
        dc.w    $0307,$fffe
        dc.w    $180,$030
        dc.w    $0507,$fffe
        dc.w    $180,$040
        dc.w    $0707,$fffe
        dc.w    $180,$050
        dc.w    $0907,$fffe
        dc.w    $180,$060
        dc.w    $0c07,$fffe
        dc.w    $180,$070
        dc.w    $0f07,$fffe
        dc.w    $180,$080
        dc.w    $1207,$fffe
        dc.w    $180,$090
        dc.w    $1607,$fffe
        dc.w    $180,$0a0
        dc.w    $1a07,$fffe
```

```
        dc.w    $180,$0b0
        dc.w    $1f07,$fffe
        dc.w    $180,$0c0
        dc.w    $2607,$fffe
        dc.w    $180,$0d0
        dc.w    $2c07,$fffe
        dc.w    $180,$0e0

coppajumpa2:
        dc.w    $80     ; COP1lc per far ripartire la copperlist dall'ourcopper
        DC.W    0
        dc.w    $82     ; COP2Lcl
        DC.W    0
        dc.w    $FFFF,$FFFE     ; Fine della copperlist
finepezzofinale:


        section bufcopperi,bss_C

copcols:
copbuf1:
        ds.b    ((linee*8)+AGGIUNTE)*NUMBUFCOPPERI      ; 50 copperlist!
finebuffoni:

        end
```

Se vi precalcolate l'effetto copper, le moltiplicazioni, le coordinate dei
3d vectors, la musica... potete fare una demo che lascia libero il processore
per fare aleno 1 effetto non precalcolato!!!! HAHAHAHA!

Da notare che dalla copper1 si salta alla copper2, alla cui fine si salta
alla copper "pezzofinale", che ripunta come copper di partenza la copper1!
Dunque facciamo saltare 2 volte il copper, e non 1 come in Lezione11h.s

Lezione11h4

```
; Lezione11h4.s BARRETTA CHE SALE E SCENDE USANDO IL MASCHERAMENTO DEL WAIT

;       Questo listato e' identico al Lezione3d.s, fatta eccezione per
;       uno stratagemma che ci permette di spostare l'intera barretta
;       con una sola istruzione!!!! Il trucco sta nella COPPERLIST!


        SECTION CiriCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
```

```
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        btst    #2,$dff016      ; tasto destro premuto?
        beq.s   Mouse2          ; se si non eseguire MuoviCopper

        bsr.s   MuoviCopper     ; Routine che sfrutta il mascheramento del WAIT

mouse2:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************

MuoviCopper:
        TST.B   SuGiu           ; Dobbiamo salire o scendere? se SuGiu e'
                                ; azzerata, (cioe' il TST verifica il BEQ)
                                ; allora saltiamo a VAIGIU, se invece e' a $FF
                                ; (se cioe' questo TST non e' verificato)
                                ; continuiamo salendo (facendo dei subq)
        beq.w   VAIGIU
        cmpi.b  #$34,BARRA      ; siamo arrivati alla linea $34?
        beq.s   MettiGiu        ; se si, siamo in cima e dobbiamo scendere
        subq.b  #1,BARRA
        rts

MettiGiu:
        clr.b   SuGiu           ; Azzerando SuGiu, al TST.B SuGiu il BEQ
        rts                     ; fara' saltare alla routine VAIGIU, e
                                ; la barra scedera'

VAIGIU:
        cmpi.b  #$77,BARRA      ; siamo arrivati alla linea $77?
        beq.s   MettiSu         ; se si, siamo in fondo e dobbiamo risalire
        addq.b  #1,BARRA
        rts

MettiSu:
        move.b  #$ff,SuGiu      ; Quando la label SuGiu non e' a zero,
        rts                     ; significa che dobbiamo risalire.
```

```
Finito:
        rts


;       Questo byte, indicato dalla label SuGiu, e' un FLAG, ossia una
;       bandierina (in gergo)

SuGiu:
        dc.b    0,0


*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200
        dc.w    $180,$000       ; Inizio la cop col colore NERO

        dc.w    $2c07,$FFFE     ; una piccola barretta fissa verde
        dc.w    $180,$010
        dc.w    $2d07,$FFFE
        dc.w    $180,$020
        dc.w    $2e07,$FFFE
        dc.w    $180,$030
        dc.w    $2f07,$FFFE
        dc.w    $180,$040
        dc.w    $3007,$FFFE
        dc.w    $180,$030
        dc.w    $3107,$FFFE
        dc.w    $180,$020
        dc.w    $3207,$FFFE
        dc.w    $180,$010
        dc.w    $3307,$FFFE
        dc.w    $180,$000

;         /\  __ __ ______ __ __  /\ Mo!
;       _// \/  ____ _  _ ____  \/ \\_
;       \(_  \  \(O/      \O)/  /  _)/
;        \/     _)/  _/\   \(_     \/
;        /_ __         ··\    ______ \
;       (    (_____/\  _____/ | | |\ \
;        \__________ \/   \_|_|_|_|_) )
;                   \ _______________/
;                    \/

BARRA:
        dc.w    $3407,$FFFE     ; aspetto la linea $79 (WAIT NORMALE!)
                                ; questo wait e' il "BOSS" dei wait
                                ; mascherati seguenti, infatti lo seguono
                                ; come degli scagnozzi: se questo wait
                                ; scende di 1, tutti i wait mascherati
                                ; sottostanti scendono di 1, eccetera.

        dc.w    $180,$300       ; inizio la barra rossa: rosso a 3

        dc.w    $00E1,$80FE     ; Questa coppia di istruzioni copper, che
        dc.w    $0007,$80FE     ; invece di terminare per $FFFE terminano
                                ; con $80FE, in pratica si possono tradurre
                                ; in: "ASPETTA LA LINEA SEGUENTE", in questo
                                ; caso la linea seguente al wait di BARRA:.
```

```
                        ; Infatti il $00E180fE aspetta la fine della
                        ; linea (al bordo destro dello schermo), il
                        ; che fa scattare il copper alla linea seguente
                        ; alla sua posizione orizzintale 0001 (il
                        ; bordo sinistro dello schermo). A questo
                        ; punto per "allineare" aspettiamo la posizione
                        ; 0007 come gli altri wait.

        dc.w    $180,$600       ; rosso a 6

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$900       ; rosso a 9

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$c00       ; rosso a 12

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$f00       ; rosso a 15 (al massimo)

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$c00       ; rosso a 12

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$900       ; rosso a 9

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$600       ; rosso a 6

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$300       ; rosso a 3

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

        dc.w    $180,$000       ; colore NERO


        dc.w    $fd07,$FFFE     ; aspetto la linea $FD
        dc.w    $180,$00a       ; blu intensita' 10
        dc.w    $fe07,$FFFE     ; linea seguente
        dc.w    $180,$00f       ; blu intensita' massima (15)
        dc.w    $FFFF,$FFFE     ; FINE DELLA COPPERLIST


        end
```

In questo esempio abbiamo risparmiato un bel po' di MOVE: Cambiando 1 solo BYTE per fotogramma facciamo scorrere una barra intera! Questo grazie al

```
"mascheramento della y del wait". In pratica l'utilita' sta nel fatto che
mettendo queste 2 wait mascherate:

        dc.w    $00E1,$80FE     ; ASPETTA LA LINEA SEGUENTE
        dc.w    $0007,$80FE     ; CON il Wait ad Y "mascherata"

Andiamo alla linea seguente all'ultimo wait $FFFE definito, e aggiungendo
altre coppie di $80fe si possono "appiccicare" al primo wait molte linee.
Nonostante tutto pero' e' uno stratagemma poco usato, in quanto ha delle
limitazioni, ad esempio non funziona per le linee superiori alla 127 ($7f)
circa. Provate infatti a cambiare la linea massima raggiungibile:

VAIGIU:
        cmpi.b  #$77,BARRA      ; siamo arrivati alla linea $77?

mettendoci un bel $f0, e noterete come superata la linea $80 la barra
si appiattisce e diventa una linea.
La coordinata Y deve andare da $00 a $7f, perche' possiamo mascherare solo 6
bit. Meglio di niente, pero'!!!

Quindi, si puo' dire che il mascheramento funziona nella parte alta dello
schermo da $00 a $7f circa, e sotto la zona NTSC, ossia dopo il $FFDF,$FFFE.
```

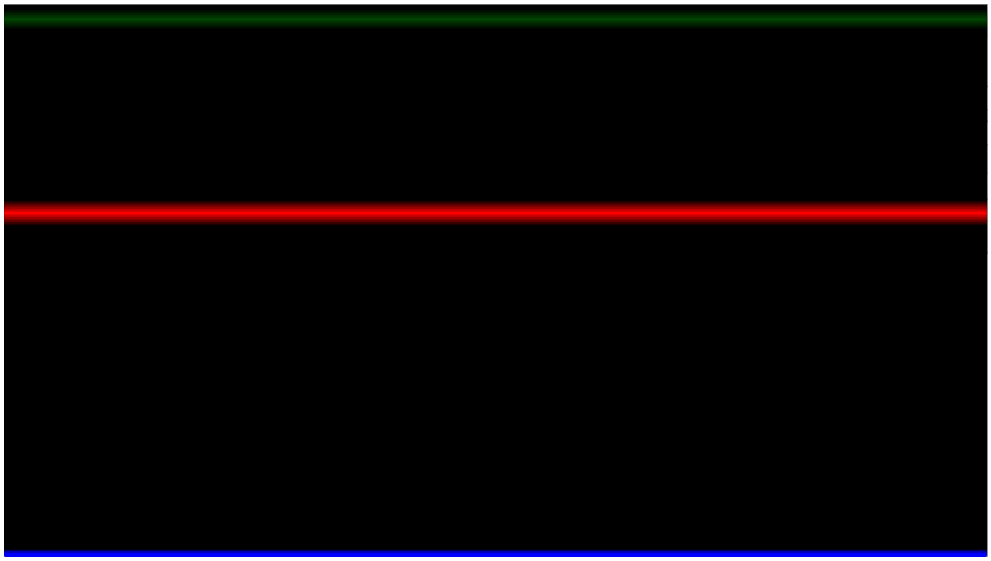


Figura 27.7: Lezione 11h4

27.9 Lezione11i1

```
; Lezione11i1.s - Scorrimento di colori a tutto schermo PAL

        SECTION Scorricol,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup2.s"            ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        BSR.w   MAKECOP         ; Fai la copperlist

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #MYCOP,$80(a5)          ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1

        btst    #2,$16(a5)      ; tasto destro premuto?
        beq.s   Mouse2          ; se si non eseguire ColorScrollPAL

        bsr.s   ColorScrollPAL          ; Scorrimento dei colori

mouse2:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************
;       Routine che crea la copperlist
*****************************************************************************

;          .__ _  __..... .   .    .
;       -\-'_ \\// _'-/-- -  -    -
;         \(-)____(-)/// /  /    /
;         -'_/V""V\_'- _  _    _
;           \ \    /__ _  _    _
;            \ \,,/__ _  _    _
;               \/-Mo!

MakeCop:
```

```
        lea     MYCOP,a0                ; Indirizzo della copperlist da fare
        move.l  #$1f07fffe,d0           ; Istruzione WAIT, prima linea $1f
                                        ; ossia WAIT di partenza
        move.l  #$0007fffe,d1           ; Ultima linea NTSC per la Copperlist
                                        ; ossia WAIT finale
        bsr.w   FaiColors               ; Fai Questo pezzo di Copperlist
                                        ; da $1f a $ff, ossia la parte NTSC

        move.l  #$ffdffffe,(a0)+        ; Wait Speciale per attendere la fine
                                        ; della zona NTSC.
        move.l  #$0007fffe,d0           ; Prima linea della zona pal (WAIT)
                                        ; ossia WAIT di partenza
        move.l  #$3707fffe,d1           ; Ultima linea in fondo allo schermo
                                        ; ossia WAIT finale
        bsr.s   FaiColors2              ; Fai il pezzo PAL della copperlist
        move.l  #$fffffffe,(a0)+        ; Fine della copperlist
        rts


*****************************************************************************
; SubRoutine che crea la copperlist - in a0 va immesso l'indirizzo della
; copperlist, in d0 il primo wait, in d1 l'ultimo da fare
*****************************************************************************

FaiColors:
        lea     ColorTabel(PC),a1       ; Indirizzo tabella colori
FaiColors2:
        move.l  d0,(a0)+                ; Immetti il WAIT in coplist
        move.w  #$0180,(a0)+            ; Immetti il registro COLOR0
        move.w  (a1)+,(a0)+             ; E il colore dalla tabella
        cmp.l   #ColorTabelEnd,a1       ; siamo all'ultimo colore della tab?
        bne.s   labelok                 ; Non ancora? allora non ripartire
        lea     ColorTabel(PC),a1       ; altrimenti riparti dal primo colore
labelok:
        addi.l  #$01000000,d0           ; Incrementa la pos Y del WAIT
        cmp.l   d0,d1                   ; Siamo arrivati all'ultimo wait?
        bne.s   FaiColors2              ; Se non ancora, fai un'altra linea
        rts


*****************************************************************************
; Routine che muove i colori
*****************************************************************************

;        \  /
;         o0
;        \__/

ColorScrollPAL:
        move.l  PuntatorecolTab(PC),a0  ; PuntatorecolTab in a0
        lea     MYCOP+6,a1              ; Indirizzo del primo colore in copper
        move.l  #225-1,d0               ; 225 colori da muovere in zona NTSC
        bsr.s   scroll                  ; Scorri la parte ntsc dello schermo
        addq.w  #4,a1                   ; salta il WAIT speciale alla fine
                                        ; della zona NTSC ($FFDFFFFE)
        moveq   #54-1,d0                ; 54 colori da muovere in zona PAL
        bsr.s   scroll                  ; Scorri la parte PAL dello schermo

        lea.l   PuntatorecolTab(PC),a0  ; PuntatorecolTab in a0
        addq.l  #2,(a0)                 ; Avanza di un colore per la prossima
                                        ; esecuzione della routine
        cmp.l   #ColorTabelEnd,(a0)     ; siamo arrivati all'ultimo colore
```

```
                                        ; della tabella??
        bne.s   NonRipartire            ; Se non ancora esci dalla routine
        move.l #ColorTabel,(a0)         ; Altrimenti riparti dall'inizio
                                        ; della tabella
NonRipartire:
        rts

*****************************************************************************
;       Subroutine che muove i colori; il numero di colori va immesso in d0,
;       l'indirizzo della tabella colori in a0 e i colori in coplist in a1
*****************************************************************************

scroll:
        move.w  (a0)+,(a1)              ; copia il colore dalla tabella alla
                                        ; copperlist
        cmp.l   #ColorTabelEnd,a0       ; Abbiamo copiato l'ultimo colore
                                        ; della tabella?
        bne.s   okay                    ; Se non ancora, continua
        lea     ColorTabel(PC),a0       ; ColorTabel in a0 - riparti dal primo
                                        ; colore della tabella
okay:
        addq.w  #8,a1                   ; Vai al prossimo colore in copperlist
        dbra    d0,scroll               ; d0 = numero colori da immettere
        rts


;       Tabella con i colori RGB

ColorTabel:
        dc.w    $000,$100,$200,$300,$400,$500,$600,$700
        dc.w    $800,$900,$a00,$b00,$c00,$d00,$e00,$f00
        dc.w    $e00,$d00,$c00,$b00,$a00,$900,$800,$700
        dc.w    $600,$500,$400,$300,$200,$100,$000,$010
        dc.w    $020,$030,$040,$050,$060,$070,$080,$090
        dc.w    $0a0,$0b0,$0c0,$0d0,$0e0,$0f0,$0e0,$0d0
        dc.w    $0c0,$0b0,$0a0,$090,$080,$070,$060,$050
        dc.w    $040,$030,$020,$010,$000,$001,$002,$003
        dc.w    $004,$005,$006,$007,$008,$009,$00a,$00b
        dc.w    $00c,$00d,$00e,$00f,$00e,$00d,$00c,$00b
        dc.w    $00a,$009,$008,$007,$006,$005,$004,$003
        dc.w    $002,$001
ColorTabelEnd:

;       Questo e' il puntatore alla tabella ColorTabel

PuntatorecolTab:
        dc.l    ColorTabel+2

*****************************************************************************
;       Copperlist creata interamente dalla routine MAKECOP; in questo modo
;       basta fare una section BSS!
*****************************************************************************

        Section Copperlist,bss_C

MYCOP:
        ds.b    225*8   ; spazio per la zona PAL
        ds.b    4       ; spazio per il $FFDFFFFE
        ds.b    55*8    ; spazio per la zona NTSC
        ds.b    4       ; spazio per la fine della copperlist $FFDFFFFE

        end
```

Figura 27.8: Lezione 11i1

Lezione11i2

```
; Lezione11i2.s - Barre in "pseudoparallasse"

        SECTION ParaCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        bsr.w   WriteWaits      ; Crea le 2 copperlist...

        lea     $dff000,a6
        MOVE.W  #DMASET,$96(a6)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #KOPLIST1,$80(a6)       ; Puntiamo la nostra COP
        move.w  d0,$88(a6)              ; Facciamo partire la COP
        move.w  #0,$1fc(a6)             ; Disattiva l'AGA
        move.w  #$c00,$106(a6)          ; Disattiva l'AGA
        move.w  #$11,$10c(a6)           ; Disattiva l'AGA

mouse:
        bsr.w   waitvb                          ; Aspetta il vertical Blank
        move.l  #koplist2,$dff080
```

```
        move.l  #koplist1Waits+6,stampa         ; inizio cop
        move.l  #koplist1Waits+6+(8*200),a5     ; fine cop
        bsr.w   cleacop                         ; pulisci la copperlist
        bsr.w   makeBeams                       ; disegna le barre

        bsr.w   waitvb                          ; Aspetta il vertical Blank
        move.l  #koplist1,$dff080
        move.l  #koplist2Waits+6,stampa         ; inzio cop
        move.l  #koplist2Waits+6+(8*200),a5     ; fine cop
        bsr.w   cleacop                         ; pulisci la copperlist
        bsr.w   makeBeams                       ; disegna le barre

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts                     ; esci

*****************************************************************************
;       Routine che crea le 2 copperlist
*****************************************************************************

;       __/\__
;       \(Oo)/
;       /_()_\
;         \/

WriteWaits:
        lea     koplist1waits,a1
        lea     koplist2waits,a2
        move.l  #$2c07ff00,d0   ; Wait (prima linea $2c)
        move.l  #$01800000,d2   ; Color0
        move.w  #200-1,d1       ; Numero waits (200 per l'area NTSC)
WWLoop:
        move.l  d0,(a1)+        ; Wait in coplist 1
        move.l  d0,(a2)+        ; Wait in coplist 2

        move.l  d2,(a1)+        ; Color0 in coplist1
        move.l  d2,(a2)+        ; Color0 in coplist2
        add.l   #$01000000,d0   ; Fai waitare 1 linea sotto
        dbra    d1,WWLoop
        RTS

*****************************************************************************
;       Routine che "pulisce" lo sfondo
*****************************************************************************

;       __/\__
;       \[oO]/
;       /_---_\
;         \/

CleaCop:
        move.l  stampa(PC),a0   ; copper attuale
        moveq   #$001,d0        ; Colore di sfondo
        move.w  #(200/4)-1,d1   ; numero waits
Clealoop:
        move.w  d0,(a0)         ; azzera
        move.w  d0,8(a0)        ;...
        move.w  d0,8*2(a0)
        move.w  d0,8*3(a0)
        lea     8*4(a0),a0
        dbra    d1,Clealoop     ; rifai 200/4 volte, perche' pulisce 4
        rts                     ; words per loop! (piu' veloce!)
```

```
*****************************************************************************
;       Routine che attende il vblank
*****************************************************************************

;       __/\__
;       \-OO-/
;       /_\/_\
;         \/

Waitvb:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$0ff00,d2      ; linea da aspettare = $FF
Waity1:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $FF
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $FF
        BEQ.S   Aspetta
        RTS

*****************************************************************************
;       Routine che modifica le copperlist
*****************************************************************************

;       __/\__
;       \(OO)/
;       /_==_\
;         \/

MakeBeams:
        lea     beam01(pc),a1   ; tab colori barra 1
        move.l  beam01x(pc),d0  ; la x...
        moveq   #10,d1          ; la distanza tra 1 e l'altro
        bsr.w   writebeam

        lea     beam02(PC),a1   ; tab colori barra 2
        move.l  beam02x(PC),d0  ; la x...
        moveq   #25,d1          ; la distanza tra 1 e l'altro
        bsr.w   writebeam

        lea     beam03(PC),a1   ; tab colori barra 2
        move.l  beam03x(PC),d0  ; la x...
        moveq   #55,d1          ; la distanza tra 1 e l'altro
        bsr.w   writebeam

; BEAM01x scende di 1 ogni 2 frames.

        subq.b  #1,timer01x     ; 1 frame ogni 2.
        bne.s   Non01x          ; passato il frame? (timer1x=0?)
        move.b  #2,timer01x     ; risetta 2 frame

        addq.l  #1,beam01x      ; Fai scendere di 1 Beam01x
        cmp.l   #8+10,beam01x   ; siamo in fondo?
        bne.s   Non01x
        clr.l   beam01x         ; Se si, riparti!
Non01x:
```

```
; BEAM02x scende di 1 ogni frame.

        addq.l  #1,beam02x      ; fai scendere di 1 beam02x
        cmp.l   #16+25,beam02x  ; siamo in fondo?
        bmi.s   NONON2
        clr.l   beam02x         ; se si riparti
NONON2:

; BEAM03x scende di 2 ogni frame.

        addq.l  #2,beam03x      ; Fai scendere di 2 beam03x

        cmp.l   #16+55,beam03x  ; siamo in fondo
        bmi.s   NONON3

        clr.l   beam03x         ; se si riparti.
NONON3:
        RTS

timer01x:
        dc.b    2

        even

stampa:         dc.l    koplist1Waits+6         ; Copper attuale

beam01x:        dc.l 10
beam02x:        dc.l 5
beam03x:        dc.l 2


*****************************************************************************
;       Routine che "scrive" le barre
*****************************************************************************
;       lea     beam01(pc),a1   ; Tabella colori Beam01
;       move.l  beam01x(pc),d0  ; la x...
;       moveq   #10,d1          ; la distanza tra 1 e l'altro

;       __/\__
;       \ $$ /
;       /_()_\
;         \/

WriteBeam:
        move.l  stampa(PC),a0   ; indirizzo copper attuale
        move.l  a1,a2           ; indirizzo tabella colori in a1 e a2
        lsl.w   #3,d0           ; X * 8
        lsl.w   #3,d1           ; Distanza tra le barre * 8
        add.w   d0,a0           ; trova offset (x*8)
WBLoop2:
        move.l  a2,a1           ; tab colori
WBLoop:
        tst.w   (a1)            ; tab colori finita?
        beq.s   EndOfBeam       ; se si, esci!
        move.w  (a1)+,(a0)      ; Copia il colore dalla tabella alla copbar
        addq.w  #8,a0           ; vai al prossimo color0
        cmp.l   a5,a0           ; fine della copperlist?
        bmi.s   WBloop          ; se non ancora, insisti
EndOfBeam:
        add.w   d1,a0           ; finita una barra, se ne fa un'altra piu'
                                ; sotto: aggiungiamo la distanza * 8 per
                                ; trovare dove comincia la prossima barra
```

```
        cmp.l   a5,a0           ; e verifichiamo di non essere fuori copper
        bmi.s   WBloop2         ; Se non siamo fuori, possiamo andare!
        RTS


; Tabella colori delle barre piu' "lontane" e lente (blu)

Beam01:
        dc.w    $003
        dc.w    $005
        dc.w    $007
        dc.w    $009
        dc.w    $00a
        dc.w    $007
        dc.w    $005
        dc.w    $003
        dc.w    0

; Tabella colori delle barre intermedie (verdi)

Beam02:
        dc.w    $001
        dc.w    $001

        dc.w    $010
        dc.w    $020
        dc.w    $030
        dc.w    $040
        dc.w    $050
        dc.w    $060
        dc.w    $070
        dc.w    $060
        dc.w    $050
        dc.w    $040
        dc.w    $030
        dc.w    $020
        dc.w    $010

        dc.w    $001
        dc.w    0

; Tabella colori delle barre "vicine" (arancione)

Beam03:
        dc.w    $110
        dc.w    $320
        dc.w    $520
        dc.w    $730
        dc.w    $940
        dc.w    $b50
        dc.w    $d60
        dc.w    $f70
        dc.w    $f60
        dc.w    $b50
        dc.w    $940
        dc.w    $730
        dc.w    $520
        dc.w    $420
        dc.w    $320
        dc.w    $210
        dc.w    $110
        dc.w    0
```

```
*****************************************************************************

        SECTION koplists,DATA_C

; Prima copper

koplist1:
        dc.w    $180,$666       ; Color0
        dc.w    $100,$200       ; bplcon0 - no bitplanes
koplist1waits:
        dcb.w   4*200,0         ; Spazio per l'effetto
        dc.w    $180,$666       ; Color0
        dc.w    $ffff,$fffe     ; Fine della copperlist

; Seconda copper, scambiata con la prima per una sorta di "double buffering"
; per copperlist, per eliminare l'evenienza di non riuscire a scrivere in
; tempo nei color0 per evitare che si "noti" la scrittura stessa.

koplist2:
        dc.w    $180,$666       ; Color0
        dc.w    $100,$200       ; bplcon0 - no bitplanes
koplist2waits:
        dcb.w   4*200,0         ; Spazio per l'effetto.
        dc.w    $180,$666       ; Color0
        dc.w    $ffff,$fffe     ; Fine della copperlist

        end

Questo listato ha la particolarita' di avere un "double coppering", ossia si
scrive su una copper mentre se ne visualizza un'altra scritta prima, in modo
da evitare che si "noti" la scrittura rallentata a video. Si poteva anche
usare il sistema del COP2LC+COPJMP2 per scambiare le copperlist.
```

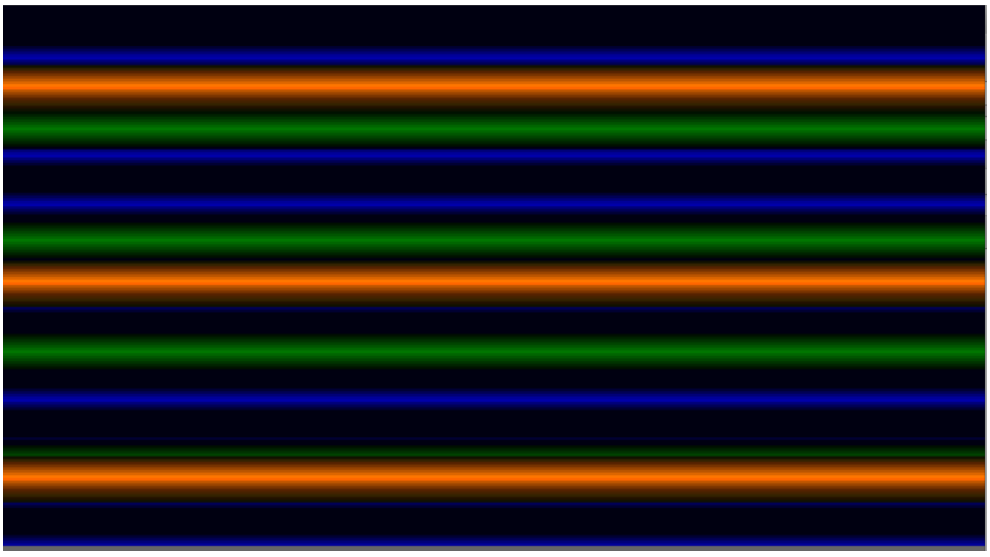


Figura 27.9: Lezione 11i2

Lezione11i3

```
; Lezione11i3.s - Fantasia in COP minore

        SECTION GnippiCop,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        bsr.w   Write           ; Crea la copperlist...

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  CopperEffPointer,$80(a5)
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $FF
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $FF
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $FF
        BEQ.S   Aspetta

        bsr.w   main

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts                     ; esci


CopperEffPointer:
        dc.l    Copperlist

colmemory:
        dc.l    colbuf



*****************************************************************************
;       Routine che crea la copperlist
```

```
*****************************************************************************

;       __/\__
;       \()()/
;       /_\/_\
;         \/

cxstart         equ     $26     ; posizione X da cui iniziare
cystart         equ     $1c     ; posizione Y da cui iniziare
ylinee          equ     280     ; numero linee Y
xlinee          equ     10      ; numero sezioni X
yDistanza       equ     1       ; distanza verticale
xDistanza       equ     20      ; distanza ORIZZONTALE tra "strisce"


write:
        move.l  CopperEffPointer(pc),a0 ; indirizzo copper
        moveq   #cystart,d0             ; posizione Y start
        move.w  #ylinee-1,d4            ; numero linee Y da fare
cr2loop:
        moveq   #cxstart,d1             ; posizione X start
        moveq   #xlinee-1,d3            ; numero blocchi orizzontali da fare
FaiOrizz:
        move.w  d1,d2           ; X in d2
        ori.b   #1,d2           ; selez. solo bit 1
        move.b  d0,(a0)+        ; WAIT - coord Y
        move.b  d2,(a0)+        ; WAIT - coord X
        move.w  #$fffe,(a0)+    ; seconda word del wait
        move.w  #$180,(a0)+     ; registro color0
        clr.w   (a0)+           ; color0 (ora azzerato)
        add.w   #xDistanza,d1   ; scatta posizione X prossima "striscia"
        dbra    d3,FaiOrizz     ; fai tutta la linea orizzontale

        addq.w  #yDistanza,d0   ; aggiungi la distanza Y
        dbra    d4,cr2loop      ; fai tutte le linee
        move.l  #$fffffffe,(a0)+ ; fine della copperlist!
        rts

*****************************************************************************
;       Routine che modifica la copperlist
*****************************************************************************

;       __/\__
;       \ oO /
;       /_<>_\
;         \/

speed           equ     6
stadd           equ     100


main:
        move.l  colmemory(pc),a0        ; tabella per i colori
        lea     pointer1(pc),a1         ; puntatore 1
        lea     addtable(pc),a2         ; tabella con valori da addare
        moveq   #0,d4
        move.w  pointer2(pc),d4         ; puntatore 2 in d4
        addq.w  #speed,d4               ; + speed
        andi.w  #$1ff,d4                ; seleziona solo 9 bit (max 511)
        move.w  d4,pointer2             ; salva in puntatore 2
        bclr.l  #8,d4                   ; azzera bit 8
        beq.s   nosub                   ; =0 (era 256?)
```

```
        move.w  #$100,d1
        sub.w   d4,d1
        move.w  d1,d4
nosub:
        add.w   #stadd,d4       ; salta 100
        moveq   #xlinee-1,d1    ; num. linee X
Cstart:
        clr.w   d0
        move.b  (a2)+,d0        ; prendi valore da addtable
        bclr.l  #7,d0
        bne.s   pstripe
        bclr.l  #6,d0
        bne.s   sub
        bclr.l  #5,d0
        bne.s   add
back:
        dbra    d1,Cstart       ; fai per tutte le linee
        bra.s   copy

*****************************************************************************

add:
        bsr.s   addr
        bra.s   back

sub:
        bsr.s   subr
        bra.s   back

sub1:
        bsr.s   subr
        bra.s   sback

add1:
        bsr.s   addr
        bra.s   sback

*****************************************************************************

;       __/\__
;       \-OO-/
;       /_-)_\
;         \/

colors          equ     210     ; num coloritab

addr:
        moveq   #0,d2
        move.w  (a1),d2
        add.w   d0,d2
        cmp.w   #colors-2,d2    ; finiti i colori tab?
        blo.s   noclr
        clr.w   d2
noclr:
        move.w  d2,(a1)+
        bra.s   do_it

*****************************************************************************

;       __/\__
;       \[oO]/
;       /_{}_\
```

```
;         \/

subr:
        moveq   #0,d2
        move.w  (a1),d2
        sub.w   d0,d2
        bpl.s   nomove
        move.w  #colors-2,d2
nomove:
        move.w  d2,(a1)+
do_it:
        lea     colortable(pc),a4
        add.l   d2,a4
        move.w  #ylinee/2-1,d2
do_loop:
        move.l  (a4)+,(a0)+     ; metti colori in coplist
        dbra    d2,do_loop
        rts

*****************************************************************************

pstripe:
        move.l  a0,d3
        bclr.l  #5,d0
        bne.s   add1
        bclr.l  #6,d0
        bne.s   sub1
clear:
        move.w  #ylinee/2-1,d2
clloop:
        clr.l   (a0)+           ; azzera tutte le linee
        dbra    d2,clloop
sback:
        move.l  d3,a0
        add.l   d4,a0
        moveq   #stlinee/4-1,d2
        lea     stable(pc),a4
csloop:
        move.l  (a4)+,(a0)+     ; copia dalla stable i colori in cop
        dbra    d2,csloop
        move.l  d3,a0
        add.l   #ylinee*2,a0
        bra.s   back

*****************************************************************************

;             :
;            _|_
;       _ __/__/\__ _
;           \__\/
;             |
;             :

copy:
        move.l  colmemory(pc),a0
        move.l  CopperEffPointer(pc),a1
        addq.w  #6,a1
        move.w  #ylinee-1,d0    ; fai tutte le linee Y
coloop:
        move.w  (a0),(a1)
        move.w  ylinee*2(a0),8(a1)      ; copia da colmemory a coplist
        move.w  ylinee*4(a0),8*2(a1)
```

```
        move.w  ylinee*6(a0),8*3(a1)
        move.w  ylinee*8(a0),8*4(a1)
        move.w  ylinee*10(a0),8*5(a1)
        move.w  ylinee*12(a0),8*6(a1)
        move.w  ylinee*14(a0),8*7(a1)
        move.w  ylinee*16(a0),8*8(a1)
        move.w  ylinee*18(a0),8*9(a1)
        lea     8*10(a1),a1
        addq.w  #2,a0
        dbra    d0,coloop
        rts

*****************************************************************************
;                       TABELLE DEI COLORI
*****************************************************************************

colortable:
        dc.w    $001,$002,$003,$004,$005,$006,$007,$008,$009,$00a
        dc.w    $00b,$00c,$00d,$00e,$00f
        dc.w    $01f,$02f,$03f,$04f,$05f
        dc.w    $06f,$07f,$08f,$09f,$0af,$0bf,$0cf,$0df,$0ef,$0ff
        dc.w    $0fe,$0fd,$0fc,$0fb,$0fa,$0f9,$0f8,$0f7,$0f6,$0f5
        dc.w    $0f4,$0f3,$0f2,$0f1,$0f0,$1f0,$2f0,$3f0,$4f0,$5f0
        dc.w    $6f0,$7f0,$8f0,$9f0,$af0,$bf0,$cf0,$df0,$ef0,$ff0
        dc.w    $fe0,$fd0,$fc0,$fb0,$fa0,$f90,$f80,$f70,$f60,$f50
        dc.w    $f40,$f30,$f20,$f10,$f00,$f01,$f02,$f03,$f04,$f05
        dc.w    $f06,$f07,$f08,$f09,$f0a,$f0b,$f0c,$f0d,$f0e,$f0f
        dc.w    $e0e,$d0d,$c0c,$b0b,$a0a,$909,$808,$707,$606
        dc.w    $505,$404,$303,$202,$101,$000
colorend:
        dc.w    $001,$002,$003,$004,$005,$006,$007,$008,$009,$00a
        dc.w    $00b,$00c,$00d,$00e,$00f
        dc.w    $01f,$02f,$03f,$04f,$05f
        dc.w    $06f,$07f,$08f,$09f,$0af,$0bf,$0cf,$0df,$0ef,$0ff
        dc.w    $0fe,$0fd,$0fc,$0fb,$0fa,$0f9,$0f8,$0f7,$0f6,$0f5
        dc.w    $0f4,$0f3,$0f2,$0f1,$0f0,$1f0,$2f0,$3f0,$4f0,$5f0
        dc.w    $6f0,$7f0,$8f0,$9f0,$af0,$bf0,$cf0,$df0,$ef0,$ff0
        dc.w    $fe0,$fd0,$fc0,$fb0,$fa0,$f90,$f80,$f70,$f60,$f50
        dc.w    $f40,$f30,$f20,$f10,$f00,$f01,$f02,$f03,$f04,$f05
        dc.w    $f06,$f07,$f08,$f09,$f0a,$f0b,$f0c,$f0d,$f0e,$f0f
        dc.w    $e0e,$d0d,$c0c,$b0b,$a0a,$909,$808,$707,$606
        dc.w    $505,$404,$303,$202,$101,$000
        dc.w    $001,$002,$003,$004,$005,$006,$007,$008,$009,$00a
        dc.w    $00b,$00c,$00d,$00e,$00f
        dc.w    $01f,$02f,$03f,$04f,$05f
        dc.w    $06f,$07f,$08f,$09f,$0af,$0bf,$0cf,$0df,$0ef,$0ff
        dc.w    $0fe,$0fd,$0fc,$0fb,$0fa,$0f9,$0f8,$0f7,$0f6,$0f5
        dc.w    $0f4,$0f3,$0f2,$0f1,$0f0,$1f0,$2f0,$3f0,$4f0,$5f0
        dc.w    $6f0,$7f0,$8f0,$9f0,$af0,$bf0,$cf0,$df0,$ef0,$ff0
        dc.w    $fe0,$fd0,$fc0,$fb0,$fa0,$f90,$f80,$f70,$f60,$f50
```

```
        dc.w    $f40,$f30,$f20,$f10,$f00,$f01,$f02,$f03,$f04,$f05
        dc.w    $f06,$f07,$f08,$f09,$f0a,$f0b,$f0c,$f0d,$f0e,$f0f
        dc.w    $e0e,$d0d,$c0c,$b0b,$a0a,$909,$808,$707,$606
        dc.w    $505,$404,$303,$202,$101,$000


pointer1:
        dcb.w   10,0

pointer2:
        dcb.w   10,0

addtable:
        dc.b    $c4,$a2,$44,$a0,$24,$42,$a0,$22,$a4,$c2

; colorsequcolorend-colortable

numctab         equ     512


stripe:
        dc.w    numctab*4

; La barra griagia che "attraversa" quelle colorate

stable:
        dc.w    $000,$111,$222,$444,$555,$666,$777,$888,$888,$999,$999
        dc.w    $aaa,$aaa,$bbb,$bbb,$ccc,$ccc,$ddd,$ddd,$eee,$eee,$eee
        dc.w    $fff,$fff,$fff,$fff,$fff,$eee,$eee,$eee,$ddd,$ddd,$ccc
        dc.w    $ccc,$bbb,$bbb,$aaa,$aaa,$999,$999,$888,$888,$777,$666
        dc.w    $555,$444,$333,$222,$111,$000
stend:

stlinee equ     stend-stable

*****************************************************************************
;                       Buffer vari
*****************************************************************************

        section bau1,bss

clsize          equ     2*xlinee*ylinee

colbuf:
        ds.b    clsize


*****************************************************************************
;                       Copperlist
*****************************************************************************

        section bau2,bss_c

csize           equ     xlinee*ylinee*8+12

Copperlist:
        ds.b    csize

        END
```

Non si direbbe che non ci sono bitplanes abilitati, eh!?

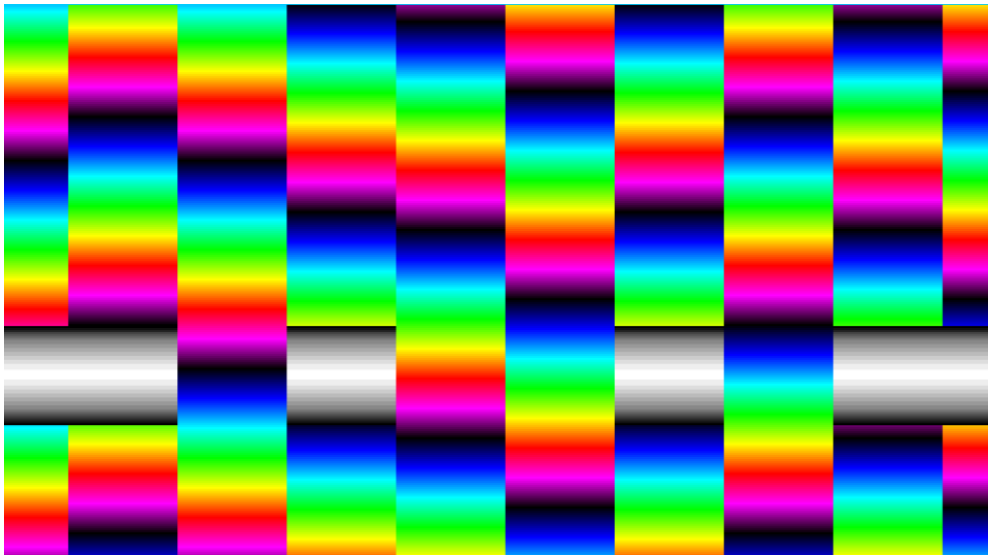


Figura 27.10: Lezione 11i3

Lezione11i4

```
;  Lezione11i4.s - Effetto originale di shadow/ELECTRON modificato.

; PREMERE IL TASTO DESTRO PER CAMBIARE TONALITA' ALLE SFUMATURE...

        SECTION Barrex,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

range           equ     20
NumeroLinee     equ     257

START:

        bsr.w   initcopbuf      ; Prepara la copperlist

        lea     $dff000,a6
        MOVE.W  #DMASET,$96(a6)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COP,$80(a6)            ; Puntiamo la nostra COP
        move.w  d0,$88(a6)              ; Facciamo partire la COP
        move.w  #0,$1fc(a6)             ; Disattiva l'AGA
        move.w  #$c00,$106(a6)          ; Disattiva l'AGA
```

```
        move.w  #$11,$10c(a6)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A6),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

        bsr.w   copmove         ; effetto main - sfumatura colori
        bsr.w   cycle           ; fa scorrere (cicare) i colori

        btst.b  #2,$dff016              ; tasto destro del mouse premuto?
        bne.s   NonCambiarMaschera
        move.w  6(a6),MascheraColori    ; VHPOSR - metti un valore a caso
        move.b  7(a6),d0                ; HPOSR
        and.w   #%011001110011,MascheraColori

NonCambiarMaschera:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        rts


*****************************************************************
; Questa routine fa "scorrere" i colori dal centro verso i bordi
*****************************************************************

Chiarostep:
        dc.w    10

cycle:
        lea     copbuf+6+8,a0           ; primo colore in alto
        lea     copbuf+6-8+[256*8],a1   ; ultimo colore in fondo

        moveq   #128-1,d0       ; numero di cicli
cycleloop:
        subq.w  #01,count       ; ogni "chiarostep" schiarisci il colore.
        bne.s   gocycle
        add.w   #$101,(a0)      ; schiariamo il colore 1 ogni 10
        move.w  ChiaroStep(PC),count
gocycle:
        move.w  (a0),-8(a0)     ; scroll verso l'alto della parte superiore
        move.w  (a0),8(a1)      ; scroll verso il basso della parte inferiore
        addq.w  #8,a0
        subq.w  #8,a1
        dbra    d0,cycleloop
        rts

count:
        dc.w    10

*****************************************************************
; Questa routine sfuma i colori
```

```
************************************************************

copmove:
        lea     copbuf+6+[128*8],a1     ; meta' schermo
smooth:
        move.w  ColoreOld(pc),d0
        move.w  ColoreNewCaso(pc),d1
        cmp.w   d0,d1                   ; colore vecchio uguale al nuovo?
        beq.s   newcol          ; allora prendi un nuovo colore "a caso"

        subq.w  #01,counter     ; counter = 0?
        beq.s   gosmooth        ; se si "sfuma"...
        bra.s   draw            ; altrimenti metti semplicemente.

; "sfumatura" dei colori - adda e subba semplicemente le componenti, nulla di
; eccezionale.

gosmooth:
        move.w  #range,counter

        move.w  d0,d2
        move.w  d1,d3
        and.w   #$000f,d2       ; solo componente bli
        and.w   #$000f,d3
        cmp.w   d2,d3
        beq.s   blueready
        bgt.s   addblue
subblue:
        sub.w   #$0001,d0       ; - blu
        bra.s   blueready
addblue:
        add.w   #$0001,d0       ; + blu
blueready:
        move.w  d0,d2
        move.w  d1,d3
        and.w   #$00f0,d2       ; solo componente verde
        and.w   #$00f0,d3
        cmp.w   d2,d3
        beq.s   greenready
        bgt.s   addgreen
subgreen:
        sub.w   #$0010,d0       ; - verde
        bra.s   greenready
addgreen:
        add.w   #$0010,d0       ; + verde
greenready:
        move.w  d0,d2
        move.w  d1,d3
        and.w   #$0f00,d2       ; solo componente rossa
        and.w   #$0f00,d3
        cmp.w   d2,d3
        beq.s   redready
        bgt.s   addred
subred:
        sub.w   #$0100,d0       ; - rosso
        bra.s   redready
addred:
        add.w   #$0100,d0       ; + rosso
redready:
        move.w  d0,ColoreOld
draw:
        move.w  d0,(a1)
```

```
        rts

;---------------------------------------------------------------------------
; Prende un colore a caso facendo casino con la posizione orizzontale del
; pennello elettronico. Non e' una gran routine ma funzionicchia per
; avere valori "pseudocasuali".
;---------------------------------------------------------------------------

newcol:
        move.w  ColoreNewCaso(pc),ColoreOld

        move.b  $05(a6),d1      ; $dff006 - per colore RANDOM...
        muls.w  #$71,d1
        eor.w   #$ed,d1
        muls.w  $06(a6),d1      ; $dff006 - per colore RANDOM
        and.w   MascheraColori(PC),d1   ; seleziona solo i bit mascheracolori
        move.w  d1,ColoreNewCaso

        cmp.w   ColoreOld(pc),d1
        bne.w   smooth
        add.b   #$08,ColoreNewCaso
        bra.w   smooth


MascheraColori:
                dc.w    $012

ColoreOld:              dc.w    0
ColoreNewCaso:  dc.w    0
counter:        dc.w    range

************************************************************ initcopbuf
;       crea la copperlist
************************************************************ initcopbuf

initcopbuf:
        lea     copbuf,a0
        move.l  #$29e1fffe,d0   ; prima linea wait

        move.w  #NumeroLinee-1,d1
coploop:
        move.l  d0,(a0)+                ; metti il wait
        move.l  #$01800000,(a0)+        ; color0
        add.l   #$01000000,d0           ; fai waitare una linea sotto
        dbra    d1,coploop
        rts

************************************************************* coplist
;                               COPPERLIST
************************************************************* coplist

        section gfx,data_C

cop:
                dc.w    $100,$200       ; bplcon0 - no bitplanes
copbuf:
                ds.b    NumeroLinee*8   ; spazio per l'effetto copper

                dc.w    $ffff,$fffe     ; Fine della copperlist
        end
```

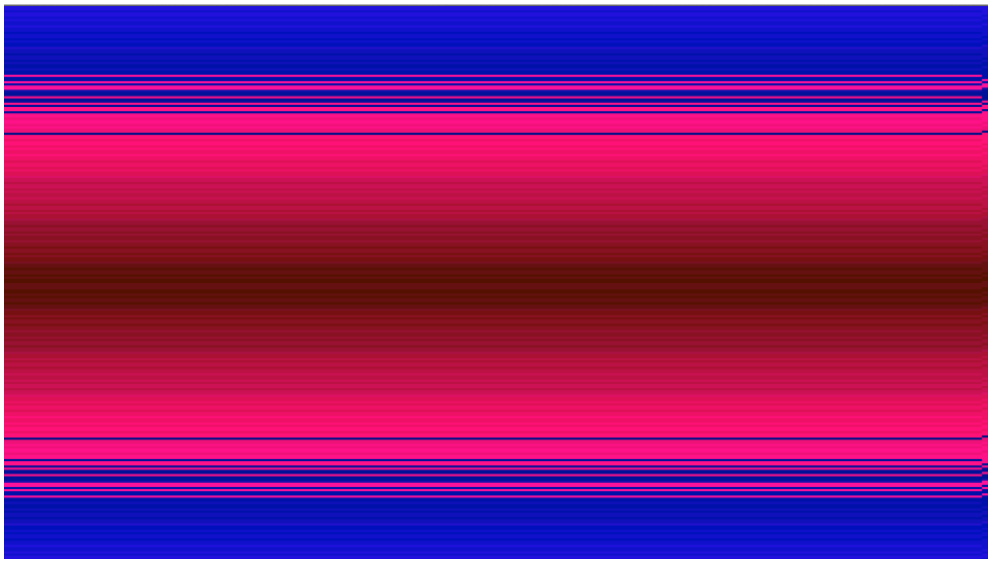


Figura 27.11: Lezione 11i4

Lezione11i5

```
;  Lezione11i5.s - Una modifica alla solita barra....

; Tasto destro per abbassare la barra; si potrebbe fare una tabella per
; farla rimbalzare in alto e in basso

        SECTION Coppex,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
```

```
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

        btst    #2,$dff016      ; tasto destro del mouse?
        bne.s   NonAbbassare
        cmp.b   #$c0,OrizzCoord ; barra gia' abbastanza bassa?
        bhi.s   NonAbbassare
        addq.b  #1,OrizzCoord

NonAbbassare:
        bsr.s   CoolRaster

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************
;       routine principale
*****************************************************************************

CoolRaster:
        ADDQ.W  #2,OrizzCoord
        BSR.S   CoolEffetto
        BSR.S   ScorriColori    ; fai scorrere i colori della tab
        rts


*****************************************************************************
;       Routine di scorrimento dei colori della parte rossa dell'effetto
;       i colori sono fatti scorrere direttamente nella ColorTab1
*****************************************************************************

ScorriColori:
        LEA     ColorTab1(PC),A0
        MOVE.W  (A0),30*2(A0)           ; salva il primo colore in fondo
        LEA     ColorTab1+2(PC),A1      ; indirizzo secondo colore
        MOVEQ   #31-1,D1                ; 30 colori da "spostare"
ScorriTAB:
        MOVE.W  (A1)+,(A0)+             ; colore 2 in colore 1, colore 3 in
        DBRA    D1,ScorriTAB            ; colore 2 eccetera.
        RTS

*****************************************************************************

;              - ----   ---- -
;                    \ /
;          .:::::::::: ::::::::::.
;       ( :::        + +        ::: )
;          `:::::::::: ::::::::::'
;              /__  /  \\  __\
;              \_\ (_____) /_/
;           _/    \_ ___ _/    \_
;           |       V   V       |
;          /|\                 /|\
```

```
;          |||                |||
;

CoolEffetto:
        LEA     CopperBuffer1,A0
        LEA     ColorTab1(PC),A1        ; tabella colori 1
        LEA     ColorTab2(PC),A2        ; tabella colori 2

        MOVEQ   #29-1,D0        ; 29 linee per l'effetto
        MOVE.W  OrizzCoord(PC),D1       ; attuale wait orizzontale e vert.in d1
WRITEBOTHLINES:
        MOVE.W  D1,(A0)+        ; mettila in copperlist
        MOVE.W  #$FFFE,(A0)+    ; seguito dal $FFFE
        MOVE.W  #$0180,(A0)+    ; Color0
        MOVE.W  (A1)+,(A0)+     ; metti il colore dalla tab1
        ADD.W   #$0020,D1       ; sposta il wait 20 passi piu' avanti
        MOVE.W  D1,(A0)+        ; e mettilo in copperlist
        MOVE.W  #$FFFE,(A0)+    ; seguito dal $FFFE
        MOVE.W  #$0180,(A0)+    ; color0
        MOVE.W  (A2)+,(A0)+     ; metti il colore dalla tab2
        ADD.W   #$0020,D1       ; sposta il wait 20 passi piu' avanti
        DBRA    D0,WRITEBOTHLINES
        RTS


;       Tabella della sfumatura rossa

ColorTab1:      ; 30 valori.w RGB per il color0 in copperlist

        dc.W    $100,$200,$300
        dc.W    $400,$500,$600,$700,$800,$900,$A00,$B00,$C00,$D00,$E00,$F00
        dc.W    $E00,$D00,$C00,$B00,$A00,$900,$800,$700,$600,$500,$400,$300
        dc.W    $200,$100,$101



;       Tabella della sfumatura grigia

ColorTab2:      ; 30 valori.w RGB per il color0 in copperlist

        dc.W    $000
        dc.W    $111,$222,$333,$444,$555,$666,$777,$888,$999,$AAA,$BBB,$CCC
        dc.W    $DDD,$EEE,$DDD,$CCC,$BBB
        dc.W    $AAA,$999,$888,$777,$666,$555,$444,$333,$222,$111,$000
        dc.w    $000

;       Questo e' il wait iniziale

OrizzCoord:
        dc.W $3A07


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $0180,$0000     ; color0 nero
        DC.W    $2B07,$FFFE     ; wait linea $2b
```

```
CopperBuffer1:
        dcb.W   29*8,0

        dc.W    $0180,$000      ; color0 nero


        dc.w    $d007,$fffe     ; Wait linea $d0
        dc.w    $180,$035
        dc.w    $d207,$fffe     ; Wait linea $d0
        dc.w    $180,$047
        dc.w    $d607,$fffe     ; Wait linea $d0
        dc.w    $180,$059

        dc.W    $FFFF,$FFFE     ; fine della copperlist

        end
```

Figura 27.12: Lezione 11i5

Lezione11i6

```
;  Lezione11i6.s - effetto sfumato copper "pseudo 3d"

        SECTION Barrex,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; solo copper DMA
```

```
WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        bsr.s   makerast        ; Fai la copperlist

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

        bsr.s   MakeRast        ; rulla i colori

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************
;       Routine che crea la copperlist
*****************************************************************************

;         Oo
;        `--'

MakeRast:
        lea Offsets(PC),a2      ; tabella con 8*20 valori degli offset tra le
                                ; linee wait
        sub.w   #1*20,ContatoreWaitAnim
        bpl.s   nocolscroll
        addq.b  #1,ContatoreColore
        move.w  #7*20,ContatoreWaitAnim
nocolscroll:
        moveq   #0,d0           ; azzera d0
        move.w  ContatoreWaitAnim(PC),d0
        add.w   d0,a2           ; trova l'offset giusto nella tabella Offsets
        lea     CopBuffer,a0

        moveq   #0,d0
        move.b  ContatoreColore(PC),d0

        moveq   #20,d3          ; numero loops FaiCopper
        lea Colors(PC),a1       ; tabella con i colori
FaiCopper:
        and.w   #%01111111,d0   ; servono solo i primi 7 bit di d0
```

```
        move.w  d0,d2           ; rimetti in d2 l'ultimo valore del colore
                                ; salvato
        asl.l   #1,d2           ; e spostalo a sinistra di 1 bit, il che
                                ; significa moltiplicare il valore per 2, dato
                                ; che i valori nella tabella sono .w (2 bytes)
                                ; in questo modo il valore di d2 e' pronto
                                ; per il "move.w (a1,d2),(a0)+" finale

        addq.b  #1,d0           ; prossimo colore per il prossimo loop

        moveq   #0,d1           ; azzera d1
        move.b  (a2)+,d1        ; prendi il prossimo offset dalla tabella

        add.b   #$0f,d1         ; offset dalla linea $00, ossia dall'inizio
                                ; dello schermo, da aggiungere ai valori
                                ; letti nella TAB
        asl.w   #8,d1           ; sposta il valore a sinistra di 8 bit,dato che
                                ; si tratta della coordinata verticale
                                ; es: prima era $0019, allora diventa $1900

        or.w    #$07,d1         ; linea orizzontale dei wait: 07 (con l'OR si
                                ; aggiunge lo 07 finale, es: $1907,$fffe...)
        move.w  d1,(a0)+        ; prima word del wait con linea e colonna
        move.w  #$fffe,(a0)+    ; seconda word del WAIT
        move.w  #$0180,(a0)+    ; COLOR0
        move.w  (a1,d2),(a0)+   ; copia il colore giusto dalla tabella alla
                                ; copperlist
        dbra    d3,FaiCopper
        rts



;       tabella con i colori della sfumatura. 128 valori.w

Colors:
        dc.w $111,$444,$222,$777,$333,$aaa,$333,$aaa    ; prima parte grigia
        dc.w $333,$aaa,$333,$aaa,$333,$aaa,$333,$aaa
        dc.w $222,$777,$222,$444,$111,$000

        dc.w $000,$100,$200,$300,$400,$500,$600,$700    ; parte colorata
        dc.w $800,$900,$a00,$b00,$c00,$d00,$e00
        dc.w $f00,$f10,$f20,$f30,$f40,$f50,$f60,$f70
        dc.w $f80,$f90,$fa0,$fb0,$fc0,$fd0,$fe0
        dc.w $ff0,$ef0,$df0,$cf0,$bf0,$af0,$9f0,$8f0
        dc.w $7f0,$6f0,$5f0,$4f0,$3f0,$2f0,$1f0
        dc.w $0f0,$0f1,$0f2,$0f3,$0f4,$0f5,$0f6,$0f7
        dc.w $0f8,$0f9,$0fa,$0fb,$0fc,$0fd,$0fe
        dc.w $0ff,$0ef,$0df,$0cf,$0bf,$0af,$09f,$08f
        dc.w $07f,$06f,$05f,$04f,$03f,$02f,$01f
        dc.w $00f,$10f,$20f,$30f,$40f,$50f,$60f,$70f
        dc.w $80f,$90f,$a0f,$b0f,$c0f,$d0f,$e0f
        dc.w $f0f,$e0e,$d0d,$c0c,$b0b,$a0a,$909,$808
        dc.w $707,$606,$505,$404,$303,$202,$101,$000


; Tabella per distanze tra una linea e l'altra.
; Sono 8 linee di 20 valori, per un totale di 20*8=160 bytes
; Da notare che mentre i primi valori di ogni linea sono molto distanti fra
; loro (0,16,28,37...) gli ultimi arrivano ad essere consecutivi (77,78,79)
; Questo e' per rendere una specie di prospettiva:
;
;       ------------------------------------------------------------
```

```
;
;       ---------------------------------------------------------
;       ---------------------------------------------------------
;       ---------------------------------------------------------
;       ---------------------------------------------------------
;
; Ci sono 8 linee di 20 valori, in quanto ogni fotogramma i wait "si spostano"
; scorrendo in alto (si noti: 0,16.. prima linea, 2,18... la seconda, 6,21 la
; terza). In questo modo, oltre ad essere disposti in "pseudo-prospettiva",
; scorrono verso l'alto rendendo l'effetto piu' credibile. Potremmo dire che
; questa e' una tabella con 8 "fotogrammi" di animazione dei wait!!!

Offsets:
        dc.b  0,16,28,37,44,50,54,58,61,64,66,68,70,72,74,75,76,77,78,79
        dc.b  2,18,29,38,45,50,55,58,61,64,66,68,70,72,74,75,76,77,78,79
        dc.b  4,20,31,39,45,51,55,58,62,64,67,69,71,72,74,75,76,77,78,79
        dc.b  6,21,32,40,46,51,56,59,62,65,67,69,71,72,74,75,76,77,78,79
        dc.b  8,23,33,41,47,52,56,60,62,65,67,69,71,72,74,75,76,77,78,79
        dc.b 10,24,34,42,48,52,56,60,63,65,68,69,71,73,74,75,76,77,78,79
        dc.b 12,25,35,42,48,53,57,60,63,66,68,70,71,73,74,75,76,77,78,79
        dc.b 14,27,36,43,49,54,57,61,63,66,68,70,71,73,74,75,76,77,78,79

ContatoreWaitAnim:
        dc.w    7*20

ContatoreColore:
        dc.b    0

        even

*****************************************************************************
;       Copperlist
*****************************************************************************

        Section Grafica,data_C

copperlist:
        dc.w    $8e,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$38         ; DdfStart
        dc.w    $94,$d0         ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,40         ; Bpl1Mod
        dc.w    $10a,40         ; Bpl2Mod

        dc.w    $180,$000       ; Color0 nero
        dc.w    $100,$200       ; bplcon0 - no bitplanes

CopBuffer:
        dcb.w   21*4,0          ; spazio dove viene creato l'effetto

        dc.w    $6007,$fffe     ; "pavimentazione" grigia
        dc.w    $0180,$0444
        dc.w    $6207,$fffe
        dc.w    $0180,$0666
        dc.w    $6507,$fffe
        dc.w    $0180,$0888
        dc.w    $6907,$fffe
        dc.w    $0180,$0aaa

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
        end
```

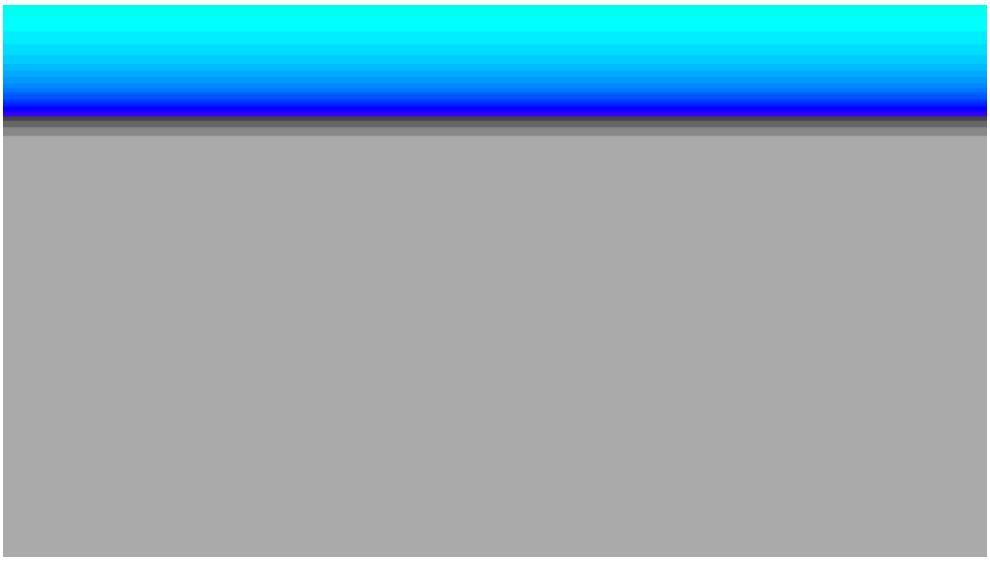


Figura 27.13: Lezione 11i6

27.10 Lezione11l1

```
;  Lezione11l1.s - cambiamo ad ogni linea il color0 e il bplcon1 ($dff102)

        SECTION coplanes,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
```

```
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 1     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)

START:
;        PUNTIAMO IL NOSTRO BITPLANE

        MOVE.L  #BITPLANE,d0
        LEA     BPLPOINTER,A1
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        move.w  #11,ContaNumLoop1
        move.w  #2,Contatore1
        clr.w   Contatore2

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

        btst.b  #2,$dff016
        beq.s   NoEff
        bsr.s   Mainroutine     ; rulla i colori e rolla il bplcon1
NoEff:
        bsr.w   PrintCarattere  ; Stampa un carattere alla volta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
```

```
        rts

*****************************************************************************
; Questa routine non e' per niente ottimizzata, si potrebbe fare una
; routine che crea la copperlist, da chiamare all'inizio, poi un'altra
; che cambia solo i valori del color1 e del bplcon1.
; Gia' che era lenta, per peggiorarla e' stato usato un sistema che
; comunque puo' servire in certi casi: per "scorrere" le tabelle, viene
; usato un buffer lungo quanto la tabella in cui viene copiata la tabella
; stessa ruotata, poi da questa tabella i valori sono ricopiati nella
; tabella di partenza. Ma non si faceva prima senza il buffer? Si!
; Ma pensate ad una routine con tante tabelle, che possono tenere i valori
; nelle varie fasi della rotazione. In questo caso potermmo "precalcolarci"
; in tante tabelle i valori ruotati in ogni fase... ma forse si otterrebbe
; cosi' poca ottimizzazione che non varrebbe la pena... insomma date un
; occhiata alla routine, e' "strana" e si ingarbuglia per niente, proprio
; per mostrare tecniche "alternative"... (esagerato.. fa schifo e basta!).
*****************************************************************************

MainRoutine:
        move.l  a5,-(sp)        ; salviamo a5
        subq.w  #1,Contatore1   ; Segna questa esecuzione
        tst.w   Contatore1      ; 2 frame passati?
        bne.w   SaltaRull       ; Se non ancora non rullare
        move.w  #2,Contatore1   ; Riparti, ad aspettare 2 frames
        cmp.w   #15,Contatore2  ; Passati 15 frames?
        beq.s   Rulla2
        addq.w  #1,ContaNumLoop1
        cmp.w   #30,ContaNumLoop1 ; siamo a 30 loops da fare?
        bne.s   VaiARullare       ; se non ancora ok
        move.w  #15,Contatore2    ; Altrimenti Contatore2=15
        bra.s   VaiARullare
Rulla2:
        subq.w  #1,ContaNumLoop1        ; subbiamo
        cmp.w   #3,ContaNumLoop1        ; siamo a 3?
        bne.s   VaiARullare             ; Se non ancora RiRulla
        clr.w   Contatore2              ; Altrimenti azzera contatore2
VaiARullare:
        lea     coltab(PC),a0   ; Tabella con i colori
        lea     TabBuf(PC),a1
        move.w  (a0)+,d0        ; Primo colore in d0
CopiaColtabLoop:
        move.w  (a0)+,d1        ; Prossimo colore in d1
        cmp.w   #-2,d1  ; fine tabella?
        beq.s   FiniTabCol      ; Se si, il lop e' finito
        move.w  d1,(a1)+        ; se no, metti questo colore nella TabBuf
        bra.s   CopiaColtabLoop

FiniTabCol:
        move.w  d0,(a1)+        ; Metti il primo colore come ultimo
        move.w  #-2,(a1)+       ; E poni il segno di fine tabella
        lea     coltab(PC),a0   ; Tab colori
        lea     TabBuf(PC),a1   ; Buffer tab
RicopiaInColTabLoop:
        move.w  (a1)+,d0        ; copia colore da TabBuf
        move.w  d0,(a0)+        ; Mettilo in coltab
        cmp.w   #-2,d0          ; Fine?
        bne.s   RicopiaInColTabLoop
SaltaRull:
        lea     BplCon1Tab(PC),a0 ; Tab con valori per bplcon1
        lea     TabBuf(PC),a1
        move.w  (a0)+,d0        ; Privo val. della tab salvato in d0
```

```
RullaLoop:
        move.w  (a0)+,d1        ; Prossimo val. tab Bplcon1
        cmp.w   #-2,d1          ; Fine tabella?
        beq.s   rullFinito      ; Se si salta avanti
        move.w  d1,(a1)+        ; Copia da BplCon1Tab a TabBuf
        bra.s   RullaLoop
rullFinito:
        move.w  d0,(a1)+        ; Metti il primo valore come ultimo
        move.w  #-2,(a1)+         ; metti flag di fine tabella
        lea     BplCon1Tab(PC),a0 ; Tab valori bplcon1
        lea     TabBuf(PC),a1     ; buffer
RicopiaCon1:
        move.w  (a1)+,d0        ; copia da tabbuf
        move.w  d0,(a0)+        ; a bplcon1tab
        cmp.w   #-2,d0          ; siamo alla fine?
        bne.s   RicopiaCon1     ; se non ancora, ricopia!
delayed:
        lea     CopperEffect,a0

; primo loop, che fa la parte ntsc (prime $ff linee)

        move.w  #$2007,d0       ; posizione wait start YY=$22
        move.w  #$4007,d2       ; posizione wait step YY=$22
        moveq   #7-1,d4         ; Numero di loops da $20 l'uno.
                                ; $20*7=$e0, + $20 iniziale = $100, ossia
                                ; tutta la zona NTSC
        lea     FineTabCol(PC),a1
        lea     BplCon1Tab(PC),a2 ; tab valori per bplcon1
loop:
        move.w  ContaNumLoop1(PC),d3
main:
        move.w  (a2)+,d5        ; Prissimo valore del bplcon1
        cmp.w   #-2,d5          ; Fine tabella?
        bne.s   initd           ; Se no, continua
        lea     BplCon1Tab(PC),a2 ; Altrimenti, riparti da capo
        move.w  (a2)+,d5        ; valore del bplcon1
initd:
        move.w  -(a1),d1        ; leggi il colore e vai indietro
        cmp.w   #-2,d1          ; Fine tabella?
        bne.s   initc           ; Se non ancora, metti il colore & bplcon1
        lea     FineTabCol(PC),a1 ; Altrimenti riparti dalla fine della tabcol
        move.w  -(a1),d1        ; leggi il colore e vai indietro
initc:
        move.w  d0,(a0)+        ; YYXX del wait
        move.w  #$fffe,(a0)+    ; wait
        move.w  #$0180,(a0)+    ; registro color0
        move.w  d1,(a0)+        ; valore del color0
        move.w  #$0102,(a0)+    ; bplcon1
        move.w  d5,(a0)+        ; valore del bplcon1
        add.w   #$0100,d0       ; fai waitare una linea sotto
        dbra    d3,main
second:
        move.w  (a2)+,d5        ; Prossimo Bplcon1val
        cmp.w   #-2,d5          ; Fine tabella?
        bne.s   doned
        lea     BplCon1Tab(PC),a2 ; riparti dall'inizio
        move.w  (a2)+,d5        ; Prossimo valore Bplcon1
doned:
        move.w  (a1)+,d1        ; Prossimo colore
        cmp.w   #-2,d1          ; Fine tabella?
        bne.s   done
        lea     coltab(PC),a1   ; riparti dall'inizio
```

```
        move.w  (a1)+,d1        ; Prossimo colore in tab
done:
        move.w  d0,(a0)+        ; YYXX del wait
        move.w  #$fffe,(a0)+    ; wait
        move.w  #$0180,(a0)+    ; registro color0
        move.w  d1,(a0)+        ; valore del color0
        move.w  #$0102,(a0)+    ; registro bplcon1
        move.w  d5,(a0)+        ; valore del reg. bplcon1
        add.w   #$0100,d0       ; fai waitare una linea sotto
        cmp.w   d2,d0           ; siamo alla fine del blocco da $20 linee?
        bne.s   second
        add.w   #$2000,d2       ; sposta il nuovo massimo $20 piu' in basso.
        dbra    d4,loop
        move.l  #$fffffffe,(a0)+        ; Fine zona ntsc

; Secondo loop, che fa la zona PAL, sotto la linea $FF

        move.w  #$0007,d0       ; Inizo wait, alla linea $00 (ossia 256)
        move.w  #$2007,d2       ; Fine alla linea $20 (+$ff)
        moveq   #2-1,d4         ; Numero loops
loop2:
        move.w  ContaNumLoop1(PC),d3
main2:
        move.w  -(a1),d1        ; colore precedente
        cmp.w   #-2,d1          ; Fine tab?
        bne.s   initc2
        lea     FineTabCol(PC),a1 ; riparti dalla fine della tabCol
        move.w  -(a1),d1        ; Colore precedente
initc2:
        move.w  d0,(a0)+        ; YYXX del wait
        move.w  #$fffe,(a0)+    ; Wait
        move.w  #$0180,(a0)+    ; registro color0
        move.w  d1,(a0)+        ; valore del color0
        add.w   #$0100,d0       ; fai waitare una linea sotto
        dbra    d3,main2
second2:
        move.w  (a1)+,d1        ; Prossimo colore
        cmp.w   #-2,d1          ; fine tab?
        bne.s   done2
        lea     coltab(PC),a1   ; Tabella colori -  riparti dall'inizio
        move.w  (a1)+,d1        ; Prossimo colore in d1
done2:
        move.w  d0,(a0)+        ; coord YYXX del wait
        move.w  #$fffe,(a0)+    ; seconda word del wait
        move.w  #$0180,(a0)+    ; registro color0
        move.w  d1,(a0)+        ; Valore del color0
        add.w   #$0100,d0       ; Fai waitare una linea sotto
        cmp.w   d2,d0           ; Siamo in fondo? ($20-$40-$60)
        bne.s   second2         ; Se non ancora, insisti
        add.w   #$2000,d2       ; Poni il massimo 20 piu' in basso
        dbra    d4,loop2
        move.l  (sp)+,a5        ; Ripristiniamo a5
        rts

ContaNumLoop1:  dc.w    0
Contatore1:     dc.w    0
Contatore2:     dc.w    0


        dc.w    -2      ; inizio tab
coltab:
        dc.w    $000,$000,$000,$000,$000,$000,$000,$000,$000,$000,$000
```

```
        dc.w    $000,$001,$002,$003,$004,$005,$006,$007,$008,$009,$009
        dc.w    $00a,$00a,$00b,$00b,$00b,$01c,$02c,$03c,$04c,$05d,$05d
        dc.w    $06d,$06d,$07d,$07d,$07d,$08d,$08d,$08d,$09d,$09D,$09C
        dc.w    $0aA,$0aA,$0a9,$0a8,$0a7,$0a6,$0a5,$0a4,$0a3,$0b2,$0b1
        dc.w    $0b0,$1b0,$2b0,$3b0,$4b0,$5b0,$6b0,$7b0,$8b0,$9b0,$Ab0
        dc.w    $Bb0,$Cb0,$Db0,$db0,$db0,$db0,$db0,$da0,$da0,$d90,$d90
        dc.w    $d80,$d70,$d60,$d50,$d40,$d30,$d20,$d10,$d00,$d00,$D00
        dc.w    $C00,$B00,$A00,$900,$800,$700,$600,$500,$400,$300,$200
        dc.w    $100,$000,$000
FineTabCol:
        dc.w    -2      ; fine tab

; Tabella dei valori per il bplcon1. Come notate si provoca un ondeggio.

        dc.w    -2      ; inizio tab
BplCon1Tab:
        dc.w    $11,$11,$11,$22,$22,$33,$44,$55,$55,$66,$66,$66,$077,$077
        dc.w    $77,$77,$77,$77,$66,$66,$66,$55,$55,$44,$33,$33,$022,$022
        dc.w    $22,$11,$11,$11,$11,$00,$00,$00,$00,$00,$00,$11,$011,$011
        dc.w    $11,$11,$22,$22,$22,$22,$33,$33,$44,$44,$55,$55,$055,$055
        dc.w    $66,$66,$66,$66,$66,$66,$77,$77,$77,$77,$77,$77,$077,$077
        dc.w    $77,$77,$66,$66,$66,$66,$66,$66,$55,$55,$55,$55,$044,$044
        dc.w    $33,$33,$33,$33,$22,$22,$22,$22,$22,$22,$11,$11,$011,$011
        dc.w    -2      ; fine tab

; In questo buffer vengono ricopiate le tabelle ruotate, che poi vengono
; ricopiate nelle tabelle stesse... un modo strano per scorrere, no?

TabBuf:
        ds.w    128

*****************************************************************************
;                       Routine di Print
*****************************************************************************

PRINTcarattere:
        movem.l d2/a0/a2-a3,-(SP)
        MOVE.L  PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        CMP.B   #$ff,d2         ; Segnale di fine testo? ($FF)
        beq.s   FineTesto       ; Se si, esci senza stampare
        TST.B   d2              ; Segnale di fine riga? ($00)
        bne.s   NonFineRiga     ; Se no, non andare a capo

        ADD.L   #40*7,PuntaBITPLANE     ; ANDIAMO A CAPO
        ADDQ.L  #1,PuntaTesto           ; primo carattere riga dopo
                                        ; (saltiamo lo ZERO)
        move.b  (a0)+,d2                ; primo carattere della riga dopo
                                        ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        LSL.W   #3,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D2,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L  PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3
```

```
                            ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,40*3(A3)  ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,40*4(A3)  ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,40*5(A3)  ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,40*6(A3)  ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,40*7(A3)  ; stampa LA LINEA 8  " "

        ADDQ.L  #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L  #1,PuntaTesto   ; prossimo carattere da stampare

FineTesto:
        movem.l (SP)+,d2/a0/a2-a3
        RTS


PuntaTesto:
        dc.l    TESTO

PuntaBitplane:
        dc.l    BITPLANE

;       $00 per "fine linea" - $FF per "fine testo"

                ; numero caratteri per linea: 40
TESTO:       ;           1111111111222222222233333333334
             ;  12345678901234567890123456789012345678901
        dc.b    '                                        ',0 ; 1
        dc.b    '    Questo listato cambia ad ogni       ',0 ; 2
        dc.b    '                                        ',0 ; 3
        dc.b    '    linea sia il color1 ($dff184),      ',0 ; 4
        dc.b    '                                        ',0 ; 5
        dc.b    '    che il bplcon1 ($dff102). Notate    ',0 ; 6
        dc.b    '                                        ',0 ; 7
        dc.b    '    come si possano "unire" listati     ',0 ; 8
        dc.b    '                                        ',0 ; 9
        dc.b    '    visti in precedenza in un solo      ',0 ; 10
        dc.b    '                                        ',0 ; 11
        dc.b    '    effetto. Si potrebbero cambiare     ',0 ; 12
        dc.b    '                                        ',0 ; 13
        dc.b    '    anche altri colori e i moduli per   ',0 ; 14
        dc.b    '                                        ',0 ; 15
        dc.b    '    ogni linea, se avete voglia         ',0 ; 16
        dc.b    '                                        ',0 ; 17
        dc.b    '    provate!                            ',$FF ; 18

        EVEN

;       Il FONT caratteri 8x8 (copiato in CHIP dalla CPU e non dal blitter,
;       per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin  "assembler2:sorgenti4/nice.fnt"

*****************************************************************************

        section graficozza,data_C

COPPERLIST:
```

```
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop
        dc.w    $100,BPLC0      ; BplCon0
        dc.w    $180,$000       ; color0 nero
        dc.w    $182,$eee       ; color1 bianco
BPLPOINTER:
        dc.w    $E0,$0000       ; Bpl0h
        dc.w    $E2,$0000       ; Bpl0l
        dc.w    $102,$0         ; Bplcon1
        dc.w    $104,$0         ; Bplcon2
        dc.w    $108,$0         ; Bpl1mod
        dc.w    $10a,$0         ; Bpl2mod

CopperEffect:
        dcb.l   801,0           ; spazio per l'effetto (attenzione! se
                                ; cambiate l'effetto puo' diventare piu'
                                ; grande o piu' piccolo)
        dc.w    $ffff,$fffe     ; Fine copperlist

*****************************************************************************

        SECTION MIOPLANE,BSS_C

BITPLANE:
        ds.b    40*256  ; un bitplane lowres 320x256

        end
```

Avrete notato che aggrovigliamento e quanti loop strani, regolati da contatori, faccia la routine. questo serve per creare quell'effeto dei colori, che non e' un semplice scorrimento in alto o in basso, ma "l'incrocio" di piu' scorrimenti, dato da vari passaggi.

Figura 27.14: Lezione 11l1

Lezione11l2

```
;  Lezione11l2.s - Si cambiano per ogni linea ben 3 colori su 4 (2 bitplanes).

        SECTION coplanes,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 2     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:

; Puntiamo i planes

        MOVE.L  #Bitplane1,d0
        LEA     PLANES,a0
        MOVEQ   #2-1,d7         ; 2 bitplanes
PLOOP:
        move.w  d0,6(a0)
        swap    d0
        move.w  d0,2(a0)
        swap    d0
        add.l   #40*256,d0
        addq.w  #8,a0
        dbra    d7,ploop
```

```
        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #LISTE,$80(a5)          ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        bsr.s   CreaCopper      ; crea la copperlist

        LEA     TESTO(PC),A0    ; testo da stampare
        LEA     BITPLANE1,A3    ; destinazione
        bsr.w   print           ; Stampa

        LEA     TESTO2(PC),A0   ; testo da stampare
        LEA     BITPLANE2,A3    ; destinazione
        bsr.w   print           ; Stampa

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

        BSR.w   RASTERMAGIC     ; copia da tabelle in copperlist
        BSR.w   CYCLEBLU        ; cicla la tab del blu (verso il basso)
        BSR.w   CYCLERED        ; cicla la tab del rosso (verso l'alto)
        BSR.w   CYCLEGREEN      ; cicla la tab del verde (verso l'alto)

        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

*****************************************************************************
; Questa routine crea la copperlist
*****************************************************************************

;                   _____
;                  /   _/
;                 /_____\
;        _        \ o.O /
;       / )_______\_-_/_________
;       \__/       _  Y  _       /__
;        _/______/   :   \_____/__ \
;              _/    :    \_  .--(_/
;             \_____:_____/.
;             /    ___   :\ :
;            /      |   :..:
;          _/       |       \
;         _\________|        \_
;        (___________l_________/_ _ _
;                    (___________))
```

```
CreaCopper:
        LEA     CopperEffyz,A1
        MOVE.l  #$180,d2        ; col 0
        MOVE.l  #$182,d3        ; col 1
        MOVE.l  #$186,d4        ; col 3
        MOVEQ   #-2,D5          ; $FFFE ; wait command
        MOVE.W  #$0100,d6       ; WAIT ADD: 0107FFFE, 0207FFFE....
        MOVE.W  #$2C07,D1       ; start line
        MOVEQ   #7-1,D7         ; Num. loops
AGAIN:
        LEA     ColTabBlu(PC),A0
        LEA     ColTabRosso(PC),A2
        LEA     ColTabVerde(PC),A3

        REPT    28              ; num of lines: 28*d7
        MOVE.W  D1,(A1)+        ; line to wait...
        MOVE.W  d5,(A1)+        ; $FFFE wait command
        MOVE.W  d2,(A1)+        ; col register 0
        MOVE.W  (A0)+,(A1)+     ; col value da tab blu
        MOVE.W  d3,(A1)+        ; col register 1
        MOVE.W  (A2)+,(A1)+     ; col value da tab rossa
        MOVE.W  d4,(A1)+        ; col register 3
        MOVE.W  (A3)+,(A1)+     ; col value da tab Verde
        ADD.W   d6,D1           ; Fai waitare una linea sotto
        ENDR

        DBRA    D7,AGAIN
        RTS

*****************************************************************************

;                    __ ___
;                   (______)
;                  __||||||__
;                 (__________)
;                   |      |
;                   _ __   |
;                  (.(._) (((
;                 (__ __   __)
;                 ./    \  |
;       magico!   `----( \ |
;                __ \____/ /\__
;               /  /__\___/__\ \
;              /   __   o  __   ì g®m


RASTERMAGIC:
        LEA     CopperEffyz,A1
        MOVEQ   #7-1,D7         ; Num. loops
AGAIN2:
        LEA     ColTabBlu(PC),A0
        LEA     ColTabRosso(PC),A2
        LEA     ColTabVerde(PC),A3

        REPT    28              ; num of lines: 28*d7
        addq.w  #2+4,a1         ; salta il wait e il col register 0
        MOVE.W  (A0)+,(A1)+     ; col value da tab blu
        addq.w  #2,a1           ; salta il col register 1
        MOVE.W  (A2)+,(A1)+     ; col value da tab rossa
        addq.w  #2,a1           ; salta il col register 3
        MOVE.W  (A3)+,(A1)+     ; col value da tab Verde
```

```
        ADD.W   d6,D1           ; Fai waitare una linea sotto
        ENDR

        DBRA    D7,AGAIN2
        RTS

*****************************************************************************

;           _))_
;         ./    \.
;         |      |
;        \_ __/  |
;        (-(--)  |
;       (__ __  (((
;        /__  \  __)
;  Ue'..  __)  \ | g®m
;        (_____/ |

CYCLEBLU:
        LEA     ColTabBlu+54(PC),A0
        LEA     ColTabBlu+52(PC),A1
        MOVE.W  ColTabBlu+54(PC),D1     ; salva l'ultimo colore

        REPT    27
        MOVE.W  (A1),(A0)       ; cycle2
        SUBQ.W  #2,A0
        SUBQ.W  #2,A1
        ENDR

        MOVE.W  D1,ColTabBlu    ; Rimetti l'ultimo
        RTS

*****************************************************************************

CYCLERED:
        LEA     ColTabRosso(PC),A0
        LEA     ColTabRosso+2(PC),A1
        MOVE.W  (A0),56(A0)

        REPT    29              ; cycle 2
        MOVE.W  (A1)+,(A0)+
        ENDR

        RTS

CYCLEGREEN:
        LEA     ColTabVerde(PC),A0
        LEA     ColTabVerde+2(PC),A1
        MOVE.W  (A0),56(A0)

        REPT    29              ; cycle 3
        MOVE.W  (A1)+,(A0)+
        ENDR

        RTS

ColTabBlu:
        DC.W    1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
        DC.W    15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0,0,0,0

ColTabRosso:
        DC.W    $100,$200,$300,$400,$500,$600,$700,$800,$900
```

```
        DC.W    $A00,$B00,$C00,$D00,$E00,$F00,$F00,$E00,$C00
        DC.W    $B00,$A00,$900,$800,$700,$600,$500,$400,$300
        DC.W    $200,$100,0,0,0,0

ColTabVerde:
        DC.W    $010,$020,$030,$040,$050,$060,$070,$080,$090
        DC.W    $0A0,$0B0,$0C0,$0D0,$0E0,$0F0,$0F0,$0E0,$0D0,$0C0
        DC.W    $0B0,$0A0,$090,$080,$070,$060,$050,$040,$030,$020
        DC.W    $010,0,0,0,0


*****************************************************************************
;       Routine che stampa caratteri larghi 8x8 pixel
*****************************************************************************

PRINT:
        MOVEQ   #23-1,D3        ; NUMERO RIGHE DA STAMPARE: 23
PRINTRIGA:
        MOVEQ   #40-1,D0        ; NUMERO COLONNE PER RIGA: 40
PRINTCHAR2:
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE
        LSL.W   #3,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE
        MOVE.L  D2,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,40*3(A3)  ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,40*4(A3)  ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,40*5(A3)  ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,40*6(A3)  ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,40*7(A3)  ; stampa LA LINEA 8  " "

        ADDQ.w  #1,A3           ; A1+1, avanziamo di 8 bit (PROSSIMO CARATTERE)

        DBRA    D0,PRINTCHAR2   ; STAMPIAMO D0 (40) CARATTERI PER RIGA

        ADD.W   #40*7,A3        ; ANDIAMO A CAPO

        DBRA    D3,PRINTRIGA    ; FACCIAMO D3 RIGHE

        RTS


            ; numero caratteri per linea: 40
TESTO:      ;           1111111111222222222233333333334
            ;  12345678901234567890123456789012345678901234567890
        dc.b    '   PRIMA RIGA (solo in testo1)          ' ; 1
        dc.b    '                                        ' ; 2
        dc.b    '     /\  /          # #                 ' ; 3
        dc.b    '    /  \/           # #                 ' ; 4
        dc.b    '                    # #                 ' ; 5
        dc.b    '        SESTA RIGA (entrambi i bitplane)' ; 6
        dc.b    '                                        ' ; 7
        dc.b    '                                        ' ; 8
        dc.b    'FABIO CIUCCI               INTERNATIONAL' ; 9
        dc.b    '                                        ' ; 10
        dc.b    '   1  4 6 89  !@ $ ^& () +| =- ]{       ' ; 11
        dc.b    '                                        ' ; 12
        dc.b    '     LA  A I G N T C  OBLITERAZIONE     ' ; 15
```

```
        dc.b    '                                       ' ; 25
        dc.b    '                                       ' ; 16
        dc.b    '  Nel mezzo del cammin di nostra vita  ' ; 17
        dc.b    '                                       ' ; 18
        dc.b    '    Mi RitRoVaI pEr UnA sELva oScuRa   ' ; 19
        dc.b    '                                       ' ; 20
        dc.b    '    CHE LA DIRITTA VIA ERA             ' ; 21
        dc.b    '                                       ' ; 22
        dc.b    '  AHI Quanto a DIR QUAL ERA...         ' ; 23
        dc.b    '                                       ' ; 24
        dc.b    '                                       ' ; 25
        dc.b    '                                       ' ; 26
        dc.b    '                                       ' ; 27

        EVEN

            ; numero caratteri per linea: 40
TESTO2:     ;           1111111111222222222233333333334
            ;   12345678901234567890123456789012345678901
        dc.b    '                                       ' ; 1
        dc.b    '  SECONDA RIGA (solo in testo2)        ' ; 2
        dc.b    '     /\  /          ##                 ' ; 3
        dc.b    '    /  \/           ##                 ' ; 4
        dc.b    '                    ##                 ' ; 5
        dc.b    '        SESTA RIGA (entrambi i bitplane)' ; 6
        dc.b    '                                       ' ; 7
        dc.b    '                                       ' ; 8
        dc.b    'FABIO        COMMUNICATION INTERNATIONAL' ; 9
        dc.b    '                                       ' ; 10
        dc.b    '   1234567 90  @#$%^&*( _+|\=-[]{}     ' ; 11
        dc.b    '                                       ' ; 12
        dc.b    '     LA PALINGENETICA  B I E A I N     ' ; 15
        dc.b    '                                       ' ; 25
        dc.b    '                                       ' ; 16
        dc.b    '  Nel       del cammin di        vita  ' ; 17
        dc.b    '                                       ' ; 18
        dc.b    '    Mi          pEr UnA       oScuRa   ' ; 19
        dc.b    '                                       ' ; 20
        dc.b    '    CHE LA         VIA ERA SMARRITA    ' ; 21
        dc.b    '                                       ' ; 22
        dc.b    '  AHI Quanto a     QUAL ERA...         ' ; 23
        dc.b    '                                       ' ; 24
        dc.b    '                                       ' ; 25
        dc.b    '                                       ' ; 26
        dc.b    '                                       ' ; 27

        EVEN

;       Il FONT caratteri 8x8 copiato in CHIP dalla CPU e non dal blitter,
;       per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin  "assembler2:sorgenti4/nice.fnt"

*****************************************************************************

        SECTION COP,DATA_C

LISTE:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
```

```
        dc.w    $94,DDFSt       ; DdfStop

        dc.w    $102,$0         ; Bplcon1
        dc.w    $104,$0         ; Bplcon2
        dc.w    $108,$0         ; Bpl1mod
        dc.w    $10a,$0         ; Bpl2mod
PLANES:
        DC.W    $E0,0,$E2,0,$E4,0,$E6,0
        dc.w    $100,BPLC0      ; Bplcon0 - 2 bitplanes lowres
        DC.W    $184,$fff       ; color2 giallo (quello fisso)
CopperEffyz:
        DCB.W   28*8*7          ; Spazio per l'effetto
        DC.W    $FFFF,$FFFE

*****************************************************************************

        SECTION BPLBUF,BSS_C

Bitplane1:
        ds.b    40*256
Bitplane2:
        ds.b    40*256

        END
```

Avete notato la bandierina italiana?? Mettiamo sempre un riconoscimento per gli stranieri! Ci fanno un baffo!

Figura 27.15: Lezione 11l2

Lezione11l3a

```
;  Lezione11l3a.s - spostiamo una pic in alto/basso - destra/sinitra
```

```
        SECTION coplanes,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 1     ; 0 = non ham / 1 = ham
scr_bpl         = 6     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:

; Puntiamo la PIC

        LEA     bplpointers,A0
        MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
        MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
        ADDQ.w  #8,A0
        ADD.L   #176*40,D0      ; lunghezza plane
        DBRA    D7,pointloop

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
```

```
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   sugiu           ; sposta in basso e in alto
        bsr.w   lefrig          ; sposta a destra e a sinistra

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; Mouse premuto?
        bne.s   mouse
        rts                     ; esci


*****************************************************************************
;       ROUTINE DEL LOGO SU GIU (fa puntare piu' in avanti o piu' indietro
;                                i bplpointers, niente di eccezionale
*****************************************************************************

;         _________________
;         \                /
;          \______________/_
;           | \____________/
;           |__  ..  _:
;           | \_____/ :
;         __‘---------’__
;       ./                \.
;       |  _            _  |
;       |  |            |  |

SuGiuFlag:
        DC.W    0

SUGIU:
        LEA     BPLPOINTERS,A1  ; prendi l'indirizzo attualmente puntato nei
        move.w  2(a1),d0        ; bitplanes e mettilo in d0
        swap    d0
        move.w  6(a1),d0

        BTST.b  #1,SuGiuFlag            ; devo andare su o giu?
        BEQ.S   GIUT

; VADO SU

SUT:
        MOVE.L  SUGIUTABP(PC),A0 ; tabella con multipli di 40 (del modulo)
        SUBQ.L  #2,SUGIUTABP     ; prendo il valore "prima"
        CMPA.L  #SUGIUTAB+4,A0
```

```
        BNE.S   NOBSTART
        BCHG.B  #1,SuGiuFlag            ; se ho finito, cambia direzione (vai in basso)
        ADDQ.L  #2,SUGIUTABP    ; per bilanciare
NOBSTART:
        BRA.s   NOBEND

; VADO GIU

GIUT:
        MOVE.L  SUGIUTABP(PC),A0 ; tabella con multipli di 40
        ADDQ.L  #2,SUGIUTABP     ; prendo il valore "dopo"
        CMPA.L  #SUGIUTABEND-4,A0
        BNE.S   NOBEND
        BCHG.B  #1,SuGiuFlag            ; se ho finito, cambia direzione
NOBEND:
        moveq   #0,d1
        MOVE.w  (A0),D1         ; valore da tabella in d1
        BTST.b  #1,SuGiuFlag
        BEQ.S   GIU
SU:
        add.l   d1,d0           ; se vado SU lo aggiungo
        BRA.S   MOVLOG
GIU:
        sub.l   d1,d0           ; se vado GIU lo sottraggo
MOVLOG:
        LEA     BPLPOINTERS,A1  ; e ripunto al nuovo indirizzo
        MOVEQ   #6-1,D1         ; num di bitplanes -1 (ham 6 bitplanes)
APOINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #176*40,d0      ; lunghrzza di un bitplane
        addq.w  #8,a1
        dbra    d1,APOINTB              ;Rifai D1 volte (D1=num of bitplanes)
NOMOVE:
        rts

SUGIUTABP:
        dc.l    SuGiuTab

; tabella col numero di bytes da saltare... naturalmente sono multipli di 40,
; ossia della lunghezza di una linea.

SuGiuTab:
        dc.w    0*40,0*40,0*40,1*40,0*40,0*40,1*40,0*40,1*40
        dc.w    0*40,0*40,1*40,0*40,1*40,0*40,1*40,1*40,0*40
        dc.w    0*40,0*40,1*40,0*40,1*40,0*40,1*40,0*40,0*40
        dc.w    0*40,1*40,1*40,0*40,1*40,0*40,1*40,1*40,1*40
        dc.w    1*40,0*40,1*40,1*40,1*40,1*40,0*40
        dc.w    1*40,0*40,1*40,0*40,1*40,0*40,0*40,1*40,0*40
        dc.w    0*40,1*40,0*40,1*40,0*40,0*40,1*40,0*40,0*40
SuGiuTabEnd:

*****************************************************************************
;       ROUTINE DEL LOGO DEST SINIST (usa il bplcon1, niente di speciale)
*****************************************************************************

DestSinFlag:
        DC.W    0

LefRig:
```

```
        BTST.b  #1,DestSinFlag  ; devo andare a destra o a sinistra?
        BEQ.S   ScrolDestra
ScrolSinitra:
        MOVE.L  LefRigTABP(PC),A0       ; tabella con valori per bplcon1
        SUBQ.L  #2,LefRigTABP           ; vado a sinistra
        CMPA.L  #LefRigTAB+4,A0         ; fine tabella?
        BNE.S   NOBSTART2               ; se non ancora, continua
        BCHG.B  #1,DestSinFlag          ; Altrimenti, cambia direzione
        ADDQ.L  #2,LefRigTABP           ; per bilanciare
NOBSTART2:
        BRA.s   NOBEND2

ScrolDestra:
        MOVE.L  LefRigTABP(PC),A0       ; tabella valori per bplcon1
        ADDQ.L  #2,LefRigTABP           ; vado a destra
        CMPA.L  #LefRigEND-4,A0         ; fine tabella?
        BNE.S   NOBEND2                 ; Se non ancora, continua
        BCHG.B  #1,DestSinFlag          ; Altrimenti cambia direzione
NOBEND2:
        MOVE.w  (A0),CON1               ; metti il valore nel bplcon1 nella
NOMOVE2:                                ; Copperlist
        rts

LefRigTABP:
        dc.l    LefRigTab

; Questi sono valori adatti al bplcon1 ($dff102) per scrollare a destra/sin.

LefRigTab:
        dc.w    0,0,0,0,0,0,0,$11,$11,$11,$11,$11
        dc.w    $22,$22,$22,$22,$22
        dc.w    $33,$33,$33
        dc.w    $44,$44
        dc.w    $55,$55,$55
        dc.w    $66,$66,$66,$66,$66
        dc.w    $77,$77,$77,$77,$77,$77,$77
        dc.w    $88,$88,$88,$88,$88,$88,$88,$88
        dc.w    $99,$99,$99,$99,$99,$99
        dc.w    $aa,$aa,$aa,$aa,$aa
        dc.w    $bb,$bb,$bb,$bb
        dc.w    $cc,$cc,$cc,$cc
        dc.w    $dd,$dd,$dd,$dd,$dd
        dc.w    $ee,$ee,$ee,$ee,$ee,$ee
        dc.w    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff
LefRigEnd:


*****************************************************************************
;               COPPERLIST:
*****************************************************************************

        Section MioCoppero,data_C

COPPERLIST:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop

        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
```

```
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero


        dc.w    $100,BPLC0      ; BplCon0 - 320*256 HAM


        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
        dc.w $1a0,$0666

        dc.w    $102    ; bplcon1
CON1:
        dc.w    0

        dc.w    $9707,$FFFE     ; wait linea $97
        dc.w    $100,$200       ; no bitplanes
        dc.w    $180,$110       ; color0
        dc.w    $9807,$FFFE     ; wait
        dc.w    $180,$120       ; color0
        dc.w    $9a07,$FFFE
        dc.w    $180,$130
        dc.w    $9b07,$FFFE
        dc.w    $180,$240
        dc.w    $9c07,$FFFE
        dc.w    $180,$250
        dc.w    $9d07,$FFFE
        dc.w    $180,$370
        dc.w    $9e07,$FFFE
        dc.w    $180,$390
        dc.w    $9f07,$FFFE
        dc.w    $180,$4b0
        dc.w    $a007,$FFFE
        dc.w    $180,$5d0
        dc.w    $a107,$FFFE
        dc.w    $180,$4a0
        dc.w    $a207,$FFFE
        dc.w    $180,$380
        dc.w    $a307,$FFFE
        dc.w    $180,$360
        dc.w    $a407,$FFFE
        dc.w    $180,$240
        dc.w    $a507,$FFFE
        dc.w    $180,$120
        dc.w    $a607,$FFFE
        dc.w    $180,$110
        DC.W    $A70F,$FFFE
        DC.W    $180,$000

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
        SECTION LOGO,CODE_C

LOGO:
        incbin  "amiet.raw"     ; 6 bitplanes * 176 lines * 40 bytes (HAM)

        END
```

Lezione11l3b

```
;  Lezione11l3b.s - allunghiamo "a ondeggio" una pic in senso verticale.

        SECTION coplanes,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

NumLinee        EQU     53      ; Numero di linee da fare nell'effetto.

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 1     ; 0 = non ham / 1 = ham
scr_bpl         = 6     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:

; Puntiamo la PIC

        LEA     bplpointers,A0
```

```
        MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
        MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
        ADDQ.w  #8,A0
        ADD.L   #176*40,D0      ; lunghezza plane
        DBRA    D7,pointloop

        bsr.s   PreparaCopEff   ; Prepara l'effetto copper

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        BSR.W   LOGOEFF2        ; "allunga" la pic usando i moduli

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; Mouse premuto?
        bne.s   mouse
        rts                     ; esci


*****************************************************************************
;               ROUTINE DI PREPARAZIONE DELL'EFFETTO COPPER                 *
*****************************************************************************

;        :        __
;        : .-----'  '-----.
;       _:-|_______________|___
;       \                     /
;        \________/\________/
;        : |  _         _  |
;        '-|  \________/  |
;          |               |
;          '----,,,,,,----'

PreparaCopEff:

; Crea la copperlist
```

```
        LEA     coppyeff1,A0    ; Indirizzo dove creare l'effetto in copperlist
        MOVE.L  #$1080000,D0    ; bpl1mod
        MOVE.L  #$10A0000,D1    ; bpl2mod
        MOVE.L  #$2E07FFFE,D2   ; wait (comincia dalla linea $2e)
        MOVE.L  #$01000000,D3   ; Valore da addare al wait ogni volta
        MOVEQ   #(NumLinee*2)-1,D7      ; 53 linee da fare
makecop1:
        MOVE.L  D2,(A0)+        ; Metti il WAIT
        MOVE.L  D0,(A0)+        ; Metti il bpl1mod
        MOVE.L  D1,(A0)+        ; Metti il bpl2mod
        ADD.L   D3,D2           ; Fai aspettare una linea piu' in basso al wait
        DBRA    D7,makecop1

; Moltiplica per il modulo i valori nella tabella, in modo da poter essere
; usati come valori da mettere nei BPL1MOD e BPL2MOD.

        LEA     tabby2,A0       ; Indirizzo tabella
        MOVE.W  #200-1,D7       ; Numero valori contenuti nella tabella
tab2mul:
        MOVE.W  (A0),D0         ; Prendi il valore dalla tabella
        MULU.W  #40,D0          ; Moltiplicalo per la lungh. di 1 linea (mod.)
        MOVE.W  D0,(A0)+        ; Rimetti il valore moltiplicato e avanza
        DBRA    D7,tab2mul
        rts


; Tabella contenente 200 valori .word, che saranno moltiplicati *40

tabby2:
        dc.w    0,1,0,0,1,0,0,0,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0
        dc.w    0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0
        dc.w    0,0,-1,0,0,0,-1,0,0,0,-1,0,0,0,-1,0,0
        dc.w    -1,0,0,-1,0,0,0,-1,0,0,-1,0,0,-1,0
        dc.w    0,-1,0,0,0,-1,0,0,-1,0,0,-1,0,0,0
        dc.w    -1,0,0,0,-1,0,0,0,-1,0,0,0,0,0,-1,0,0
        dc.w    0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0
        dc.w    0,0,1,0,0,1,0,0,1,0,0,0,1,0,0,1,0
tab2end:

*****************************************************************************
;                         ROUTINE COPPER EFFECT DEL LOGO                     *
*****************************************************************************

;                 _
;              .--' '-.
;              |     .|
;              |  /\__|
;              '------'____ __
;             ./ (_________|__)
;             |          |_|__)
;             |          |   __
;             |_________|  |  |
;        _____/\_     '----'  |
;       |       (______________|
;       |   _______/
;       l__|g®m

; L'effetto in copperlist e' cosi' strutturato:
;
```

```
;       DC.W    $2e07,$FFFE     ; wait
;       dc.w    $108            ; registro bpl1mod
;COPPEREFFY:
;       DC.w    xxx             ; valore bpl1mod
;       dc.w    $10A,xxx        ; registro e valore bpl1mod
;       wait... eccetera.

LOGOEFF2:
        LEA     coppyeff1+6,A0          ; Indirizzo copper effect bpl1mod
        LEA     TABBY2POINTER(PC),A4    ; Indirizzo puntatore alla tabella
        LEA     tab2end(PC),A3          ; Indirizzo fine della tabella
        MOVE.L  TABBY2POINTER(PC),A1    ; Dove siamo attualmente in tabella
        MOVEQ   #10,D0
        MOVEQ   #(NumLinee*2)-1,D7      ; numero di linee per l'effetto
LOGOEFFLOOP:
        MOVE.W  (A1),(A0)+      ; Copia il valore bpl1mod dalla tab alla cop
        MOVE.W  (A1)+,2(A0)     ;  "       "       bpl2mod       "       "
        ADDA.L  D0,A0           ; Vai al prossimo $dff108 (bpl1mod) in coplist
        CMPA.L  A3,A1           ; Era l'ultimo valore della tabella?
        BNE.S   norestart       ; Se non ancora, non ripartire
        LEA     tabby2(PC),A1   ; Altrimenti, riparti!
norestart:
        DBRA    D7,LOGOEFFLOOP
        ADDQ.L  #4,(A4)         ; Salta 2 valori in coplist (se si mette #2 si
                                ; "rallenta" l'effetto facendo leggere tutti
                                ; i 200 valori della tabella).
        CMPA.L  (A4),A3         ; Fine della tabella?
        BNE.S   NOTABENDY       ; Se non ancora, ok
        MOVE.L  #tabby2,(A4)    ; Altrimenti ripunta da capo
NOTABENDY:
        RTS

; Puntatore alla tabella usato per leggerne i valori

TABBY2POINTER:
        dc.l    tabby2

*****************************************************************************
;               COPPERLIST:
*****************************************************************************

        Section MioCoppero,data_C

COPPERLIST:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero
```

```
        dc.w    $100,BPLC0      ; BplCon0 - 320*256 HAM

        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
        dc.w $1a0,$0666

        dc.w    $102    ; bplcon1
CON1:
        dc.w    0

coppyeff1:
        dcb.w   12*NumLinee

        dc.w    $9707,$FFFE     ; wait linea $97
        dc.w    $100,$200       ; no bitplanes
        dc.w    $180,$110       ; color0
        dc.w    $9807,$FFFE     ; wait
        dc.w    $180,$120       ; color0
        dc.w    $9a07,$FFFE
        dc.w    $180,$130
        dc.w    $9b07,$FFFE
        dc.w    $180,$240
        dc.w    $9c07,$FFFE
        dc.w    $180,$250
        dc.w    $9d07,$FFFE
        dc.w    $180,$370
        dc.w    $9e07,$FFFE
        dc.w    $180,$390
        dc.w    $9f07,$FFFE
        dc.w    $180,$4b0
        dc.w    $a007,$FFFE
        dc.w    $180,$5d0
        dc.w    $a107,$FFFE
        dc.w    $180,$4a0
        dc.w    $a207,$FFFE
        dc.w    $180,$380
        dc.w    $a307,$FFFE
        dc.w    $180,$360
        dc.w    $a407,$FFFE
        dc.w    $180,$240
        dc.w    $a507,$FFFE
        dc.w    $180,$120
        dc.w    $a607,$FFFE
        dc.w    $180,$110
        DC.W    $A70F,$FFFE
        DC.W    $180,$000

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


        SECTION LOGO,CODE_C

LOGO:
        incbin  "amiet.raw"     ; 6 bitplanes * 176 lines * 40 bytes (HAM)

        END
```

Lezione11l3c

```
;  Lezione11l3c.s - ondeggiamo con movimento "vivo" una pic.

        SECTION coplanes,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

NumLinee        EQU     53      ; Numero di linee da fare nell'effetto.

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 1     ; 0 = non ham / 1 = ham
scr_bpl         = 6     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:

; Puntiamo la PIC

        LEA     bplpointers,A0
        MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
        MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
        ADDQ.w  #8,A0
        ADD.L   #176*40,D0      ; lunghezza plane
```

```
        DBRA    D7,pointloop

        bsr.s   PreparaCopEff   ; Prepara l'effetto copper

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        BSR.W   LOGOEFF2        ; "allunga" la pic usando i moduli
        bsr.w   sugiu           ; sposta in basso e in alto
        bsr.w   lefrig          ; sposta a destra e a sinistra

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; Mouse premuto?
        bne.s   mouse
        rts                     ; esci


*****************************************************************************
;               ROUTINE DI PREPARAZIONE DELL'EFFETTO COPPER                 *
*****************************************************************************

;       _____________
;       \          __/____
;        \_______________/
;         |_''__''_|
;       __(___)(___)__
;       \__  (__)  __/
;         /  ____  \
;         \ (____) /
;          \______/ g®m

PreparaCopEff:

; Crea la copperlist

        LEA     coppyeff1,A0    ; Indirizzo dove creare l'effetto in copperlist
        MOVE.L  #$1080000,D0    ; bpl1mod
        MOVE.L  #$10A0000,D1    ; bpl2mod
        MOVE.L  #$2E07FFFE,D2   ; wait (comincia dalla linea $2e)
        MOVE.L  #$01000000,D3   ; Valore da addare al wait ogni volta
        MOVEQ   #(NumLinee*2)-1,D7      ; 53 linee da fare
```

```
makecop1:
        MOVE.L  D2,(A0)+        ; Metti il WAIT
        MOVE.L  D0,(A0)+        ; Metti il bpl1mod
        MOVE.L  D1,(A0)+        ; Metti il bpl2mod
        ADD.L   D3,D2           ; Fai aspettare una linea piu' in basso al wait
        DBRA    D7,makecop1

; Moltiplica per il modulo i valori nella tabella, in modo da poter essere
; usati come valori da mettere nei BPL1MOD e BPL2MOD.

        LEA     tabby2,A0       ; Indirizzo tabella
        MOVE.W  #200-1,D7       ; Numero valori contenuti nella tabella
tab2mul:
        MOVE.W  (A0),D0         ; Prendi il valore dalla tabella
        MULU.W  #40,D0          ; Moltiplicalo per la lungh. di 1 linea (mod.)
        MOVE.W  D0,(A0)+        ; Rimetti il valore moltiplicato e avanza
        DBRA    D7,tab2mul
        rts


; Tabella contenente 200 valori .word, che saranno moltiplicati *40

tabby2:
        dc.w    0,1,0,0,1,0,0,0,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0
        dc.w    0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0
        dc.w    0,0,-1,0,0,0,-1,0,0,0,-1,0,0,0,-1,0,0
        dc.w    -1,0,0,-1,0,0,0,-1,0,0,-1,0,0,-1,0
        dc.w    0,-1,0,0,0,-1,0,0,-1,0,0,-1,0,0,0
        dc.w    -1,0,0,0,-1,0,0,0,-1,0,0,0,0,0,-1,0,0
        dc.w    0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        dc.w    0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0
        dc.w    0,0,1,0,0,1,0,0,1,0,0,0,1,0,0,1,0
tab2end:

*****************************************************************************
;                       ROUTINE COPPER EFFECT DEL LOGO                      *
*****************************************************************************

;               _
;            .--' '-.
;            |.  _  |
;            |___/  |
;            '------'
;          ./  _  \.
;        _  |   | | |
;       | |_|___| |_|
;       |     | |_||
;       |_____| (_)|
;             |    |
;            (_____|g@m
;

; L'effetto in copperlist e' cosi' strutturato:
;
;       DC.W    $2e07,$FFFE     ; wait
;       dc.w    $108            ; registro bpl1mod
;COPPEREFFY:
;       DC.w    xxx             ; valore bpl1mod
;       dc.w    $10A,xxx        ; registro e valore bpl1mod
;       wait... eccetera.
```

```
LOGOEFF2:
        LEA     coppyeff1+6,A0          ; Indirizzo copper effect bpl1mod
        LEA     TABBY2POINTER(PC),A4    ; Indirizzo puntatore alla tabella
        LEA     tab2end(PC),A3          ; Indirizzo fine della tabella
        MOVE.L  TABBY2POINTER(PC),A1    ; Dove siamo attualmente in tabella
        MOVEQ   #10,D0
        MOVEQ   #(NumLinee*2)-1,D7      ; numero di linee per l'effetto
LOGOEFFLOOP:
        MOVE.W  (A1),(A0)+      ; Copia il valore bpl1mod dalla tab alla cop
        MOVE.W  (A1)+,2(A0)     ;  "      "       bpl2mod       "       "
        ADDA.L  D0,A0           ; Vai al prossimo $dff108 (bpl1mod) in coplist
        CMPA.L  A3,A1           ; Era l'ultimo valore della tabella?
        BNE.S   norestart       ; Se non ancora, non ripartire
        LEA     tabby2(PC),A1   ; Altrimenti, riparti!
norestart:
        DBRA    D7,LOGOEFFLOOP
        ADDQ.L  #4,(A4)         ; Salta 2 valori in coplist (se si mette #2 si
                                ; "rallenta" l'effetto facendo leggere tutti
                                ; i 200 valori della tabella).
        CMPA.L  (A4),A3         ; Fine della tabella?
        BNE.S   NOTABENDY       ; Se non ancora, ok
        MOVE.L  #tabby2,(A4)    ; Altrimenti ripunta da capo
NOTABENDY:
        RTS

; Puntatore alla tabella usato per leggerne i valori

TABBY2POINTER:
        dc.l    tabby2

*****************************************************************************
;       ROUTINE DEL LOGO SU GIU (fa puntare piu' in avanti o piu' indietro
;                                i bplpointers, niente di eccezionale
*****************************************************************************

SuGiuFlag:
        DC.W    0

SUGIU:
        LEA     BPLPOINTERS,A1  ; prendi l'indirizzo attualmente puntato nei
        move.w  2(a1),d0        ; bitplanes e mettilo in d0
        swap    d0
        move.w  6(a1),d0

        BTST.b  #1,SuGiuFlag            ; devo andare su o giu?
        BEQ.S   GIUT

; VADO SU

SUT:
        MOVE.L  SUGIUTABP(PC),A0 ; tabella con multipli di 40 (del modulo)
        SUBQ.L  #2,SUGIUTABP     ; prendo il valore "prima"
        CMPA.L  #SUGIUTAB+4,A0
        BNE.S   NOBSTART
        BCHG.B  #1,SuGiuFlag            ; se ho finito, cambia direzione (vai in basso)
        ADDQ.L  #2,SUGIUTABP    ; per bilanciare
NOBSTART:
        BRA.s   NOBEND

; VADO GIU

GIUT:
```

```
        MOVE.L  SUGIUTABP(PC),A0 ; tabella con multipli di 40
        ADDQ.L  #2,SUGIUTABP     ; prendo il valore "dopo"
        CMPA.L  #SUGIUTABEND-4,A0
        BNE.S   NOBEND
        BCHG.B  #1,SuGiuFlag            ; se ho finito, cambia direzione
NOBEND:
        moveq   #0,d1
        MOVE.w  (A0),D1         ; valore da tabella in d1
        BTST.b  #1,SuGiuFlag
        BEQ.S   GIU
SU:
        add.l   d1,d0           ; se vado SU lo aggiungo
        BRA.S   MOVLOG
GIU:
        sub.l   d1,d0           ; se vado GIU lo sottraggo
MOVLOG:
        LEA     BPLPOINTERS,A1  ; e ripunto al nuovo indirizzo
        MOVEQ   #6-1,D1         ; num di bitplanes -1 (ham 6 bitplanes)
APOINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #176*40,d0      ; lunghrzza di un bitplane
        addq.w  #8,a1
        dbra    d1,APOINTB              ;Rifai D1 volte (D1=num of bitplanes)
NOMOVE:
        rts

SUGIUTABP:
        dc.l    SuGiuTab

; tabella col numero di bytes da saltare... naturalmente sono multipli di 40,
; ossia della lunghezza di una linea.

SuGiuTab:
        dc.w    0*40,0*40,0*40,1*40,0*40,0*40,1*40,0*40,1*40
        dc.w    0*40,0*40,1*40,0*40,1*40,0*40,1*40,1*40,0*40
        dc.w    0*40,0*40,1*40,0*40,1*40,0*40,1*40,0*40,0*40
        dc.w    0*40,1*40,1*40,0*40,1*40,0*40,1*40,1*40,1*40
        dc.w    1*40,0*40,1*40,1*40,1*40,1*40,0*40
        dc.w    1*40,0*40,1*40,0*40,1*40,0*40,0*40,1*40,0*40
        dc.w    0*40,1*40,0*40,1*40,0*40,0*40,1*40,0*40,0*40
SuGiuTabEnd:

*****************************************************************************
;       ROUTINE DEL LOGO DEST SINIST (usa il bplcon1, niente di speciale)
*****************************************************************************

DestSinFlag:
        DC.W    0

LefRig:
        BTST.b  #1,DestSinFlag  ; devo andare a destra o a sinistra?
        BEQ.S   ScrolDestra
ScrolSinitra:
        MOVE.L  LefRigTABP(PC),A0       ; tabella con valori per bplcon1
        SUBQ.L  #2,LefRigTABP           ; vado a sinistra
        CMPA.L  #LefRigTAB+4,A0         ; fine tabella?
        BNE.S   NOBSTART2               ; se non ancora, continua
        BCHG.B  #1,DestSinFlag          ; Altrimenti, cambia direzione
        ADDQ.L  #2,LefRigTABP           ; per bilanciare
```

```
NOBSTART2:
        BRA.s   NOBEND2

ScrolDestra:
        MOVE.L  LefRigTABP(PC),A0       ; tabella valori per bplcon1
        ADDQ.L  #2,LefRigTABP           ; vado a destra
        CMPA.L  #LefRigEND-4,A0         ; fine tabella?
        BNE.S   NOBEND2                 ; Se non ancora, continua
        BCHG.B  #1,DestSinFlag          ; Altrimenti cambia direzione
NOBEND2:
        MOVE.w  (A0),CON1               ; metti il valore nel bplcon1 nella
NOMOVE2:                                ; Copperlist
        rts

LefRigTABP:
        dc.l    LefRigTab

; Questi sono valori adatti al bplcon1 ($dff102) per scrollare a destra/sin.

LefRigTab:
        dc.w    0,0,0,0,0,0,0,$11,$11,$11,$11,$11
        dc.w    $22,$22,$22,$22,$22
        dc.w    $33,$33,$33
        dc.w    $44,$44
        dc.w    $55,$55,$55
        dc.w    $66,$66,$66,$66,$66
        dc.w    $77,$77,$77,$77,$77,$77,$77
        dc.w    $88,$88,$88,$88,$88,$88,$88,$88
        dc.w    $99,$99,$99,$99,$99,$99
        dc.w    $aa,$aa,$aa,$aa,$aa
        dc.w    $bb,$bb,$bb,$bb
        dc.w    $cc,$cc,$cc,$cc
        dc.w    $dd,$dd,$dd,$dd,$dd
        dc.w    $ee,$ee,$ee,$ee,$ee,$ee
        dc.w    $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff
LefRigEnd:

*****************************************************************************
;               COPPERLIST:
*****************************************************************************

        Section MioCoppero,data_C

COPPERLIST:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero
```

```
        dc.w    $100,BPLC0      ; BplCon0 - 320*256 HAM

        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
        dc.w $1a0,$0666

        dc.w    $102    ; bplcon1
CON1:
        dc.w    0

coppyeff1:
        dcb.w   12*NumLinee

        dc.w    $9707,$FFFE     ; wait linea $97
        dc.w    $100,$200       ; no bitplanes
        dc.w    $180,$110       ; color0
        dc.w    $9807,$FFFE     ; wait
        dc.w    $180,$120       ; color0
        dc.w    $9a07,$FFFE
        dc.w    $180,$130
        dc.w    $9b07,$FFFE
        dc.w    $180,$240
        dc.w    $9c07,$FFFE
        dc.w    $180,$250
        dc.w    $9d07,$FFFE
        dc.w    $180,$370
        dc.w    $9e07,$FFFE
        dc.w    $180,$390
        dc.w    $9f07,$FFFE
        dc.w    $180,$4b0
        dc.w    $a007,$FFFE
        dc.w    $180,$5d0
        dc.w    $a107,$FFFE
        dc.w    $180,$4a0
        dc.w    $a207,$FFFE
        dc.w    $180,$380
        dc.w    $a307,$FFFE
        dc.w    $180,$360
        dc.w    $a407,$FFFE
        dc.w    $180,$240
        dc.w    $a507,$FFFE
        dc.w    $180,$120
        dc.w    $a607,$FFFE
        dc.w    $180,$110
        DC.W    $A70F,$FFFE
        DC.W    $180,$000

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


        SECTION LOGO,CODE_C

LOGO:
        incbin  "amiet.raw"     ; 6 bitplanes * 176 lines * 40 bytes (HAM)

        END

Questa e' una fusione di Lezione11l3a.s e Lezione11l3b.s, che porta al "noto"
effetto del logo nella intro del disco 1.
```

```
Come vedete e' abbastanza semplice, nonostante faccia pensare a chissa' quali
routines.
Da notare che la SuGiuTab deve essere composta di multipli di 40 come la
tabella "tabby2": mentre tabby2 ha valori come 0, 1, -1 che poi sono
moltiplicati per 40 da una routine, SuGiuTab ta i valori gia' moltiplicati
per 40, nella forma 1*40, 2*40 eccetera. Potreste anche in quel caso mettere
solo i valori non moltiplicati, e moltiplicarli per la lunghezza di una linea,
ossia il modulo, tramite una routine. In questo modo potreste agilmente
definire un EQU:

LunghLinea      EQU     40

Per cui se voleste ondeggiare, ad esempio, una figura in hires, basterebbe
cambiare l'EQU in 80, e i valori nelle tabelle sarebbero opportunamente
moltiplicati.
```

Figura 27.16: Lezione 11l3c

Lezione11l4

```
; Lezione11l4.s  - Ondeggio della figura ottenuto cambiando ogni linea i
;                  puntatori ai bitplanes, in piu' la sfumatura del color0
;                  scorre verso l'alto.

        Section BITPLANEolljelly,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere
```

```
                ;5432109876543210
DMASET  EQU     %1000001111000000       ; copper,bitplane,blitter DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 1     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:
        bsr.s   SetCop          ; Crea la copperlist

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPER,$80(a5)         ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BNE.S   Waity1

        bsr.w   PrintCarattere  ; Stampa un carattere alla volta
        BSR.w   SistemaCop      ; Copia i valori dalle tabelle alla cop
        BSR.W   RoteaTabOndegg  ; Rotea i valori della tabella di ondeggio
        BSR.W   RoteaTabColori  ; Rotea la tabella dei colori

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$13000,d2      ; linea da aspettare = $130, ossia 304
Aspetta:
```

```
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $130 (304)
        BEQ.S   Aspetta

        btst    #6,$bfe001      ; Mouse premuto?
        bne.s   mouse
        rts                     ; esci

;******************************************************************************
; Questa routine crea la copperlist e ci immette i primi valori
;******************************************************************************

SETCOP:
        LEA     COPPER1,A0      ; Indirizzo effetto copper
        MOVE.L  ADRCOL1(PC),A2  ; Puntatore alla tab colori
        MOVE.L  #$2c07FFFE,D7   ; Wait (prima linea $30)
        MOVE.L  #BITPLANE,D0    ; Indirizzo del bitplane
        LEA     TABOSC(PC),A1   ; Tabella ondeggiamento
        MOVEQ   #39-1,D5        ; Numero valori di tabella usabili per questo
                                ; effetto. (nota: allora non e' facile capire
                                ; quante linee in pratica e' lungo l'effetto,
                                ; perche'occorre calcolare che ognuno di questi
                                ; loop puo' ripetere la linea piu' volte.
FaiEffetto:
        MOVE.B  (A1)+,D6        ; Metti prossimo valore ondeggio in d6
        TST.B   D6              ; dobbiamo tagliare gia' la linea?
        BNE.S   RipuntaLinea    ; Se no, puntiamola...
        ADDI.L  #40,D0          ; Oppure adda la lunghezza di 1 linea - punta
                                ; alla linea seguente del bitplane
        DBRA    D5,FaiEffetto   ; E continua il loop
        BRA.w   FineEffetto

RipuntaLinea:
        MOVE.L  D7,(A0)+        ; Metti il Wait in coplist
        SWAP    D0              ; swappa l'indirizzo del plane
        MOVE.W  #$E0,(A0)+      ; BPL1PTH
        MOVE.W  D0,(A0)+        ; punta la word alta
        SWAP    D0              ; swappa ancora l'indirizzo del plane
        MOVE.W  #$E2,(A0)+      ; BPL1PTL
        MOVE.W  D0,(A0)+        ; punta la word bassa
        TST.W   (A2)            ; fine tab colori?
        BNE.S   SETCOP2         ; Se non ancora, ok
        MOVE.L  ADRCOL2(PC),A2  ; Altrimenti: tab colori -> riparti
SETCOP2:
        MOVE.W  #$180,(A0)+     ; registro color0
        MOVE.W  (A2)+,(A0)+     ; valore del color0
        ADDI.L  #$01000000,D7   ; Fai waitare una linea sotto
        BCC.S   SETCOP3         ; Siamo arrivati a $FF? Se non ancora ok,
        MOVE.L  #$FFDFFFFE,(A0)+ ; Altrimenti fine zona ntsc ($FF)
        MOVE.L  #$0011FFFE,D7    ; E occorre mettere questi 2 wait.
SETCOP3:
        SUBQ.B  #1,D6   ; Subba il valore di "ripetizione linea" preso da
                        ; TABOSC.
        TST.B   D6      ; Abbiamo ripetuto abbastanza volte la linea?
        BNE.S   RipuntaLinea    ; Se non ancora, ripuntala, ripetendola

        ADDI.L  #40,D0          ; Altrimenti puntiamo piu' in basso di 1 linea
        DBRA    D5,FaiEffetto   ; e vediamo di continuare l'effetto.

FineEffetto:
        MOVE.L  #$01000200,(A0)+        ; Metti bplcon0 = no bitplanes
```

```
        MOVE.L  #$FFFFFFFE,(A0)+        ; Metti la fine della copperlist
        RTS

;*****************************************************************************
; Questa routine rotea i colori proprio nella tabella colori!
;*****************************************************************************

;        ______________
;        \    \__/    /
;         \__________/
;         __|______|__
;       __(\___)(___/)__
;       \_\    \/    /_/
;          \ \____/ /
;           \______/ g®m
;

RoteaTabColori:
        LEA     COLORSTAB(PC),A0        ; tabella colori
        MOVE.W  (A0)+,D0                ; Salva il primo colore in d0
RoteaTabColori2:
        TST.W   (A0)            ; fine della tabella?
        BNE.S   RoteaTabColori1         ; Se non ancora, ok
        MOVE.W  D0,-2(A0)       ; altrimenti metti il primo colore come ultimo
        RTS

RoteaTabColori1:
        MOVE.W  (A0)+,-4(A0)    ; Sposta (rotea) il colore "indietro"
        BRA.S   RoteaTabColori2

;*****************************************************************************
; Questa routine rotea i valori nella tabella "TABOSC"
;*****************************************************************************

RoteaTabOndegg:
        LEA     TABOSC(PC),A0   ; Indirizzo tabella
        MOVEQ   #63-1,D7        ; Numero di valori nella tabella
        MOVE.B  (A0),D0         ; Salva il primo valore in d0
RoteaTabOndegg1:
        MOVE.B  1(A0),(A0)+     ; sposta i valori "indietro".
        DBRA    D7,RoteaTabOndegg1
        MOVE.B  D0,-1(A0)       ; Rimetti il primo valore come ultimo
        RTS

;*****************************************************************************

ADRCOL1:
        DC.L    COLORSTAB
ADRCOL2:
        DC.L    COLORSTAB

COLORSTAB:
        DC.W    $FC9,$EC9,$DC9,$CC9,$CB9,$CA9,$C99,$C9A,$C9B,$C9C
        DC.W    $C9D,$C9E,$C9F,$B9F,$A9F,$99F,$9AF,$9BF,$9CF,$ACF
        DC.W    $BCF,$CCF,$DCF,$ECF,$FCF,$FBF,$FAF,$F9F,$F9E,$F9D
        DC.W    $F9C,$E9C,$D9C,$C9C,$CAC,$CBC,$CCC,$CDC,$CEC,$CFC
        DC.W    $CFB,$CFA,$CF9,$BF9,$AF9,$9F9,$9FA,$9FB,$9FC,$AFC
        DC.W    $BFC,$CFC,$DFC,$EFC,$FFC,$FEC,$FDC,$FCC,$FCB,$FCA
        DC.W    0       ; con lo zero si termina la tabella

;*****************************************************************************
; La routine non e' altro che SETCOP senza le parti che scrivono i registri
```

```
; e i wait: si scrive solo il necessario.
; Questa routine agisce sulla copperlist che ridefinisce ad ogni linea i
; puntatori ai bitplanes. Leggendo da una tabella, sa di ogni linea della
; pic quante volte ripeterla, ossia ripuntarla. Se per esempio nella tabella
; ci sono i valori 1,2,3, allora puntera' la prima linea nella prima linea
; dello schermo (1 volta), poi puntera' la seconda linea 2 volte, e la terza
; linea 3 volte. Ecco un "disegnino":
;
; linea1
; linea2
; linea2
; linea3
; linea3
; linea3
;
; Notate che la fugura si allunga...
;*****************************************************************************

SistemaCop:
        LEA     COPPER1,A0      ; Indirizzo effetto copper
        MOVE.L  ADRCOL1(PC),A2  ; Puntatore alla tab colori
        MOVE.L  #$2c07FFFE,D7   ; Wait (prima linea $30)
        MOVE.L  #BITPLANE,D0    ; Indirizzo del bitplane
        LEA     TABOSC(PC),A1   ; Tabella ondeggiamento
        MOVEQ   #39-1,D5        ; Numero valori di tabella usabili per questo
                                ; effetto. (nota: allora non e' facile capire
                                ; quante linee in pratica e' lungo l'effetto,
                                ; perche'occorre calcolare che ognuno di questi
                                ; loop puo' ripetere la linea piu' volte.
FaiEffetto2:
        MOVE.B  (A1)+,D6        ; Metti prossimo valore ondeggio in d6
        TST.B   D6              ; dobbiamo tagliare gia' la linea?
        BNE.S   RipuntaLinea2   ; Se no, puntiamola...
        ADDI.L  #40,D0          ; Oppure adda la lunghezza di 1 linea - punta
                                ; alla linea seguente del bitplane
        DBRA    D5,FaiEffetto2  ; E continua il loop
        BRA.w   FineEffetto2

RipuntaLinea2:
        addq.w  #6,a0           ; Salta il WAIT e il BPL1PTH
        SWAP    D0              ; swappa l'indirizzo del plane
        MOVE.W  D0,(A0)+        ; punta la word alta
        SWAP    D0              ; swappa ancora l'indirizzo del plane
        addq.w  #2,a0           ; salta il BPL1PTL
        MOVE.W  D0,(A0)+        ; punta la word bassa
        TST.W   (A2)            ; fine tab colori?
        BNE.S   SETCOP22        ; Se non ancora, ok
        MOVE.L  ADRCOL2(PC),A2  ; Altrimenti: tab colori -> riparti
SETCOP22:
        addq.w  #2,a0           ; salta il registro color0
        MOVE.W  (A2)+,(A0)+     ; valore del color0
        ADDI.L  #$01000000,D7   ; Fai waitare una linea sotto
        BCC.S   SETCOP32        ; Siamo arrivati a $FF? Se non ancora ok,
        addq.w  #4,a0           ; Salta l'FFDFFFFE
        MOVE.L  #$0011FFFE,D7    ; E occorre mettere questi 2 wait.
SETCOP32:
        SUBQ.B  #1,D6   ; Subba il valore di "ripetizione linea" preso da
                        ; TABOSC.
        TST.B   D6      ; Abbiamo ripetuto abbastanza volte la linea?
        BNE.S   RipuntaLinea2   ; Se non ancora, ripuntala, ripetendola

        ADDI.L  #40,D0          ; Altrimenti puntiamo piu' in basso di 1 linea
```

```
        DBRA    D5,FaiEffetto2  ; e vediamo di continuare l'effetto.

FineEffetto2:
        RTS

;*****************************************************************

; Tab con 64 valori .byte. Indica per quante linee occorre ripetere la stessa
; linea. Per esempio, dove c'e' un valore 2, la linea e' ripetura 2 volte,
; ossia e' raddoppiata in altezza.

TABOSC:
        DC.B    1,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9
        DC.B    9,9,8,8,7,7,6,6,5,5,4,4,3,3,2,2
        DC.B    2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9
        DC.B    9,9,8,8,7,7,6,6,5,5,4,4,3,3,2,1

        EVEN

*****************************************************************************
;                       Routine di Print
*****************************************************************************

PRINTcarattere:
        movem.l d2/a0/a2-a3,-(SP)
        MOVE.L  PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        CMP.B   #$ff,d2         ; Segnale di fine testo? ($FF)
        beq.s   FineTesto       ; Se si, esci senza stampare
        TST.B   d2              ; Segnale di fine riga? ($00)
        bne.s   NonFineRiga     ; Se no, non andare a capo

        ADD.L   #40*7,PuntaBITPLANE     ; ANDIAMO A CAPO
        ADDQ.L  #1,PuntaTesto           ; primo carattere riga dopo
                                        ; (saltiamo lo ZERO)
        move.b  (a0)+,d2                ; primo carattere della riga dopo
                                        ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        LSL.W   #3,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D2,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L  PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,40(A3)    ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,40*2(A3)  ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,40*3(A3)  ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,40*4(A3)  ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,40*5(A3)  ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,40*6(A3)  ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,40*7(A3)  ; stampa LA LINEA 8  " "

        ADDQ.L  #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
```

```
        ADDQ.L  #1,PuntaTesto   ; prossimo carattere da stampare

FineTesto:
        movem.l (SP)+,d2/a0/a2-a3
        RTS


PuntaTesto:
        dc.l    TESTO

PuntaBitplane:
        dc.l    BITPLANE

;       $00 per "fine linea" - $FF per "fine testo"

                ; numero caratteri per linea: 40
TESTO:       ;           1111111111222222222233333333334
             ;  12345678901234567890123456789012345678890
        dc.b    '  * * * * * * * * * * * * * * * * * *     ',0 ; 1
        dc.b    '  * MAMMA MIA MI BALLA            *     ',0 ; 2
        dc.b    '  *                    LO SCHERMO *     ',0 ; 3
        dc.b    '  * * * * * * * * * * * * * * * * * *     ',$FF ; 4

        EVEN

;       Il FONT caratteri 8x8 copiato in CHIP dalla CPU e non dal blitter,
;       per cui puo' stare anche in fast ram. Anzi sarebbe meglio!

FONT:
        incbin  "assembler2:sorgenti4/nice.fnt"

;*****************************************************************
;                               COPPERLIST
;*****************************************************************

        section cooppera,data_C

COPPER:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop
        dc.w    $100,BPLC0      ; BplCon0
        dc.w    $108,0          ; bpl1mod
        dc.w    $10a,0          ; bpl2mod
        DC.w    $182,$000       ; Color1 (scritte) - NERO
COPPER1:
        DCB.b   4000,0  ; Attenzione! La lunghezza dell'effetto dipende
                        ; dalla tabella TABOSC e non e' facile calcolarla...
        DC.L    $FFFFFFFE

;*****************************************************************
;       Il bitplane
;*****************************************************************
        section bitplane,bss_C

BITPLANE:
        ds.b    40*320

        end
```

Abbiamo visto prima un effetto simile fatto cambiando i moduli, ora cambiando

```
invece i bplpointers. Questo sistema e' piu' lento di quello con i moduli se
si devono cambiare ogni linea i puntatori di molti bitplanes, ma ogni plane
potrebbe essere definito in maniera diversa per andare per i fatti suoi,
invcece il bplmod coinvolge tutti plane pari e/o dispari.
Una particolarita' di questo sorgente e' che i valori delle tabelle per i plane
e dei colori non sono "roteati" rileggendoli dalle cop e spostandoli, ma
roteando i valori nelle tabelle stesse, per cui basta copiare ogni volta dalla
tabella alla copperlist, dopo che la tabella e' stata "roteata".
Questo sistema e' piu' veloce di altri quando si possiede fast ram ,in quanto
se si dovesse leggere da copperlist il valore e riscriverlo piu' avanti o
indietro, dovremmo accedere 2 volte alla CHIP RAM, con i relativi "ritardi",
mentre nel nostro caso accediamo alla tabella in FAST, con perdita di tempo
minima, e scriviamo solo una volta per colore/plane in CHIP. Su computer come
A4000 l'unico rallentamento e' dato dalla lettura/scrittura in CHIP RAM,
dunque la velocita' dell'esecuzione della routine raddoppia.
```

Figura 27.17: Lezione 10g3

Lezione11l5

```
; Lezione11l5.s   - "Zoom" di un'animazione che misura solo 40*29 pixel.
;                    La risoluzione finale e' 320*232, ossia 8 volte tanto.

        Section ZoomaPer8,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
```

```
DMASET  EQU     %1000001110000000       ; copper,bitplane DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 3     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:
        move.l  #planexpand,d0  ; bitplanebuffer
        LEA     BPLPOINTERS,A0
        MOVE.W  #3-1,D7         ; Numero planes
PointAnim:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        ADDQ.W  #8,A0
        SWAP    D0
        ADDI.L  #40*29,D0       ; lunghezza del bitplane di 1 frame
        DBRA    D7,PointAnim

        bsr.w   FaiCopallung    ; Fai la copperlist che allunga *8 coi moduli

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPER,$80(a5)         ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11500,d2      ; linea da aspettare = $115
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $115
```

```
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $115
        BEQ.S   Waity2

        bsr.w   CambiaFrame     ; Espandi orizzontalmente il frame attuale
                                ; di 8 volte: in pratica ogni bit diventa
                                ; un byte.

        btst    #6,$bfe001      ; Mouse premuto?
        bne.s   mouse
        rts                     ; esci

*****************************************************************************
; Routine che esegue "ZoomaFrame" ogni 7 fotogrammi, per rallentare
*****************************************************************************

CambiaFrame:
        addq.b  #1,WaitFlag
        cmp.b   #7,WaitFlag     ; Sono passati 7 frames? (per rallentare)
        bne.s   NonOra
        clr.b   WaitFlag
        bsr.w   ZoomaFrame      ; Se si, "espandiamo" il prossimo frame!
NonOra:
        rts

WaitFlag:
        dc.w    0

*****************************************************************************
; "espansione" delle pic: viene testato ogni bit, e a seconda se quest'ultimo
; e' settato o azzerato, viene immesso un byte $FF o $00.
; Da notare il BYTELOOP, che e' il centro del programma: un byte deve essere
; trasformato in 8 bytes, per cui ogni bit del byte deve essere trasformato
; in un byte. Come fare? Basta fare un btst di ognuno degli 8 bit, e ogni
; volta muovere il byte $00 o $ff a seconda dell'esito del test.
; Viene usato il registro d1 come contatore del loop dbra, ma anche come
; contatore del numero di bit da testare col btst.
*****************************************************************************

;       ___________
;       \       _  /
;        \ oO  / /
;         \\__/ /
;          \___/____  .--*
;            \______--'
;             |    |
;            _|   _|...g®m ...

ZoomaFrame:
        move.l  AnimPointer(PC),A0 ; Fotogramma piccolo attuale (40*29)
        lea     Planexpand,A1      ; Buffer destinazione (per 320*29)
        MOVE.W  #(5*29*3)-1,D7     ; 5 bytes a linea * 29 linee * 3 bitplanes
Animloop:
        moveq   #0,d0
        move.b  (A0)+,d0        ; Prossimo byte in d0
        MOVEQ   #8-1,D1         ; 8 bit da controllare e espandere.
BYTELOOP:
        BTST.l  D1,d0           ; Testa il bit del loop attuale
        BEQ.S   bitclear        ; E' azzerato?
```

```
        ST.B    (A1)+           ; Se no, setta il byte (=$FF)
        BRA.S   bitset
bitclear:
        clr.B   (A1)+           ; Se e' azzerato, azzera il byte
bitset:
        DBRA    D1,BYTELOOP     ; Controlla ed espandi tutti i bit del byte:
                                ; D1, calando, ogni volta fa fare il btst di
                                ; un bit diverso, dal 7 allo 0.

        DBRA    D7,Animloop     ; Converti tutto il fotogramma

        add.l   #(5*29)*3,AnimPointer   ; Punta al prossimo fotogramma
        move.l  AnimPointer(PC),A0
        lea     FineAnim(PC),a1
        cmp.l   a0,a1                   ; Era l'ultimo fotogramma?
        bne.s   NonRiparti
        move.l  #cannoanim,AnimPointer  ; Se si, ripartiamo dal primo
NonRiparti:
        rts

AnimPointer:
        dc.l    cannoanim

*****************************************************************************
; Routine che crea la copperlist che allunga la pic di 8 volte, usando i
; moduli in questo modo: waita una linea, poi mette i moduli a 0, in modo
; che si scatti alla linea dopo, poi riwaita la linea sotto e mette il
; modulo a -40, in modo che la stessa linea venga "replicata" ogni linea
; sotto. Dopo 7 linee waita, mette il modulo a 0 per una linea, facendo
; scattare a quella sotto, poi rimette il modulo a -40 per altre 7 linee
; per replicarla. Il risultato e' che ogni linea e' ripetuta 8 volte.
*****************************************************************************

FaiCopallung:
        lea     AllungaCop,a0   ; Buffer in copperlist
        move.l  #$3407fffe,d0   ; wait start
        move.l  #$1080000,d1    ; bpl1mod 0
        move.l  #$10a0000,d2    ; bpl2mod 0
        move.l  #$108FFD8,d3    ; bpl1mod -40
        move.l  #$10aFFD8,d4    ; bpl1mod -40
        moveq   #28-1,d7        ; numero di loops
FaiCoppa:
        move.l  d0,(a0)+        ; wait1
        move.l  d1,(a0)+        ; bpl1mod = 0
        move.l  d2,(a0)+        ; bpl2mod = 0
        add.l   #$01000000,d0   ; salta 1 linea
        move.l  d0,(a0)+        ; wait2
        move.l  d3,(a0)+        ; bpl1mod = -40
        move.l  d4,(a0)+        ; bpl2mod = -40
        add.l   #$07000000,d0   ; salta 7 linee
        cmp.l   #$0407fffe,d0   ; Siamo sotto $ff?
        bne.s   NonPAl
        move.l  #$ffdffffe,(a0)+ ; per accedere alla zona pal
NonPal:
        dbra    d7,FaiCoppa
        move.l  d0,(a0)+        ; wait finale
        rts


*****************************************************************************
; ANIMAZIONE: 8 fotogrammi larghi 40*29 pixel, a 8 colori (3 bitplanes)
*****************************************************************************
```

```
; Animazione. ogni frame misura 40*29 pixel, 3 bitplanes. Tot. 8 frames

cannoanim:
        incbin  "frame1"        ; 40*29 a 3 bitplanes (8 colori)
        incbin  "frame2"
        incbin  "frame3"
        incbin  "frame4"
        incbin  "frame5"
        incbin  "frame6"
        incbin  "frame7"
        incbin  "frame8"
FineAnim:

*****************************************************************************
;                       COPPERLISTOZZA
*****************************************************************************

        Section Copper,DATA_C

COPPER:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop

        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "

; 8 Colori

        dc.w    $180,$000,$182,$080,$184,$8c6
        dc.w    $186,$c20,$188,$d50,$18a,$e80,$18c,$0fb0
        dc.w    $18e,$ff0

        dc.w    $2c07,$FFFE     ; wait

        dc.w    $100,BPLC0      ; bplcon0 - 3 planes

        dc.w    $108,-40        ; modulo negativo - ripeti stessa linea!
        dc.w    $10A,-40
AllungaCop:
        ds.b    6*4*28          ; 2 wait + 4 move = 6*4 bytes * 21 loops
                                ; Questa copperlist allunga *8 cio' che e'
                                ; visualizzato, usando i moduli 0 e -40
                                ; alternati ogni 8 linee.
        ds.b    4*2             ; Per il $ffdffffe e per l'ultimo wait

        dc.w    $100,$200       ; bplcon0 - no bitplanes
        dc.w    $FFFF,$FFFE     ; Fine copperlist

*****************************************************************************
; Buffer dove viene "espanso" ogni fotogramma.
*****************************************************************************
```

```
        SECTION BitPlanes,BSS_C

PLANEXPAND:                     ; Dove viene espanso ogni fotogramma.
        ds.b    40*29*3         ; 40 bytes * 29 linee * 3 bitplanes

        end
```

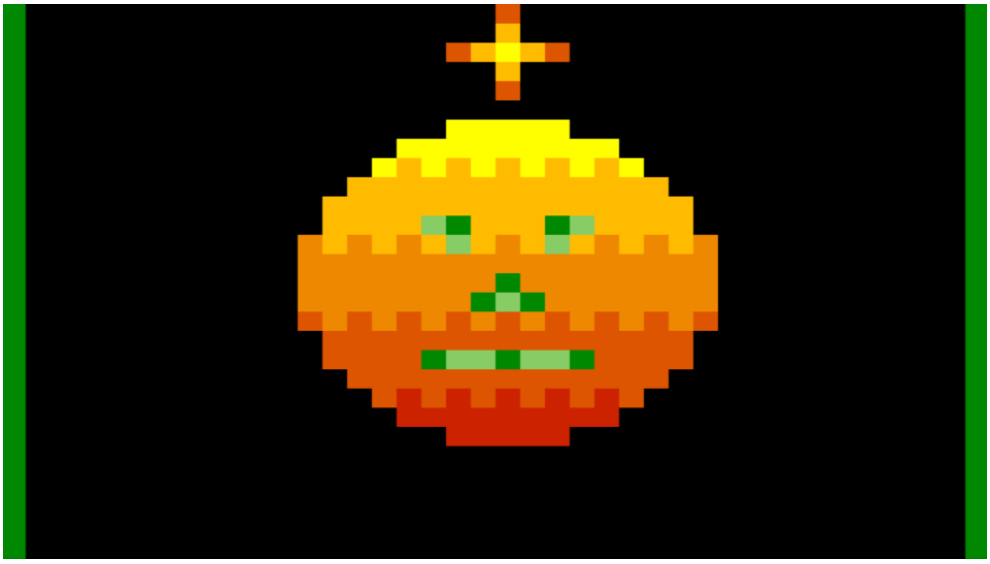


Figura 27.18: Lezione 11l5

Lezione11l5b

```
; Lezione11l5b.s   - "Zoom" di un'animazione che misura solo 40*29 pixel.
;                    La risoluzione finale e' 320*232, ossia 8 volte tanto.
;                    VERSIONE OTTIMIZZATA MEDIANTE L'USO DEL TABELLAMENTO!

        Section ZoomaPer8,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper,bitplane DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
```

```
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 0     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 3     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:
        move.l  #planexpand,d0  ; bitplanebuffer
        LEA     BPLPOINTERS,A0
        MOVE.W  #3-1,D7         ; Numero planes
PointAnim:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        ADDQ.W  #8,A0
        SWAP    D0
        ADDI.L  #40*29,D0       ; lunghezza del bitplane di 1 frame
        DBRA    D7,PointAnim

        bsr.w   PrecalcoTabba   ; Fai tab turbo con bytes "espansi" precalcol.

        bsr.w   FaiCopallung    ; Fai la copperlist che allunga *8 coi moduli

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPER,$80(a5)         ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11500,d2      ; linea da aspettare = $115
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $115
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $115
        BEQ.S   Waity2
```

```
        bsr.w   CambiaFrame     ; Espandi orizzontalmente il frame attuale
                                ; di 8 volte: in pratica ogni bit diventa
                                ; un byte.

        btst    #6,$bfe001      ; Mouse premuto?
        bne.s   mouse
        rts                     ; esci

*****************************************************************************
; Routine che esegue "ZoomaFrame" ogni 7 fotogrammi, per rallentare
*****************************************************************************

CambiaFrame:
        addq.b  #1,WaitFlag
        cmp.b   #7,WaitFlag     ; Sono passati 7 frames? (per rallentare)
        bne.s   NonOra
        clr.b   WaitFlag
        bsr.w   ZoomaFrame      ; Se si, "espandiamo" il prossimo frame!
NonOra:
        rts

WaitFlag:
        dc.w    0

*****************************************************************************
; "espansione" delle pic: viene testato ogni bit, e a seconda se quest'ultimo
; e' settato o azzerato, viene immesso un byte $FF o $00.
; Questa e' una versione OTTIMIZZATA che fa uso di una tabella che contiene
; i valori "espansi" per ogni byte possibile (uno dei 256 possibili).
*****************************************************************************

;           .-----.
;       ___ '-/\-' ____
;       \_ \_/  \_/ __/
;         \___  ___/
;           _(_\/_)_
;           \  '  /
;            \''''/
;            :\  /:
;            ! \/ !
;             _|1_

ZoomaFrame:
        move.l  AnimPointer(PC),A0 ; Fotogramma piccolo attuale (40*29)
        lea     Planexpand,A1      ; Buffer destinazione (per 320*29)
        MOVE.W  #(5*29*3)-1,D7     ; 5 bytes a linea * 29 linee * 3 bitplanes
Animloop:
        moveq   #0,d0
        move.b  (A0)+,d0        ; Prossimo byte in d0
        lsl.w   #3,d0           ; d0*8 per trovare il valore nella tabba
                                ; (ossia l'offset dal suo inizio)
        lea     Precalctabba,a2
        lea     0(a2,d0.w),a2   ; In a2 l'indirizzo nella tabba degli 8 byte
                                ; giusti per "l'espansione" degli 8 bit.
        move.l  (a2)+,(a1)+     ; 4 bytes espansi
        move.l  (a2),(a1)+      ; 4 bytes espansi (totale 8 bytes!!)

        DBRA    D7,Animloop     ; Converti tutto il fotogramma

        add.l   #(5*29)*3,AnimPointer   ; Punta al prossimo fotogramma
        move.l  AnimPointer(PC),A0
```

```
        lea     FineAnim(PC),a1
        cmp.l   a0,a1                   ; Era l'ultimo fotogramma?
        bne.s   NonRiparti
        move.l  #cannoanim,AnimPointer  ; Se si, ripartiamo dal primo
NonRiparti:
        rts

AnimPointer:
        dc.l    cannoanim

*****************************************************************************
; Routine che crea la copperlist che allunga la pic di 8 volte, usando i
; moduli in questo modo: waita una linea, poi mette i moduli a 0, in modo
; che si scatti alla linea dopo, poi riwaita la linea sotto e mette il
; modulo a -40, in modo che la stessa linea venga "replicata" ogni linea
; sotto. Dopo 7 linee waita, mette il modulo a 0 per una linea, facendo
; scattare a quella sotto, poi rimette il modulo a -40 per altre 7 linee
; per replicarla. Il risultato e' che ogni linea e' ripetuta 8 volte.
*****************************************************************************

FaiCopallung:
        lea     AllungaCop,a0   ; Buffer in copperlist
        move.l  #$3407fffe,d0   ; wait start
        move.l  #$1080000,d1    ; bpl1mod 0
        move.l  #$10a0000,d2    ; bpl2mod 0
        move.l  #$108FFD8,d3    ; bpl1mod -40
        move.l  #$10aFFD8,d4    ; bpl1mod -40
        moveq   #28-1,d7        ; numero di loops
FaiCoppa:
        move.l  d0,(a0)+        ; wait1
        move.l  d1,(a0)+        ; bpl1mod = 0
        move.l  d2,(a0)+        ; bpl2mod = 0
        add.l   #$01000000,d0   ; salta 1 linea
        move.l  d0,(a0)+        ; wait2
        move.l  d3,(a0)+        ; bpl1mod = -40
        move.l  d4,(a0)+        ; bpl2mod = -40
        add.l   #$07000000,d0   ; salta 7 linee
        cmp.l   #$0407fffe,d0   ; Siamo sotto $ff?
        bne.s   NonPAl
        move.l  #$ffdffffe,(a0)+ ; per accedere alla zona pal
NonPal:
        dbra    d7,FaiCoppa
        move.l  d0,(a0)+        ; wait finale
        rts

*****************************************************************************
; Routine che precalcola tutti i possibili 8 bytes abbinati ai possibili
; 8 bit. Per tutti si intende $FF, ossia 255.
*****************************************************************************

PrecalcoTabba:
        lea     Precalctabba,a1 ; Destinazione
        moveq   #0,d0           ; Parti dal valore zero
FaiTabba:
        MOVEQ   #8-1,D1         ; 8 bit da controllare e espandere.
BYTELOOP:
        BTST.l  D1,d0           ; Testa il bit del loop attuale
        BEQ.S   bitclear        ; E' azzerato?
        ST.B    (A1)+           ; Se no, setta il byte (=$FF)
        BRA.S   bitset
bitclear:
        clr.B   (A1)+           ; Se e' azzerato, azzera il byte
```

```
bitset:
        DBRA    D1,BYTELOOP     ; Controlla ed espandi tutti i bit del byte:
                                ; D1 calando ogni volta fa fare il btst di
                                ; tutti i bit.
        ADDQ.W  #1,D0           ; Prossimo valore
        CMP.W   #256,d0         ; Li abbiamo fatti tutti? (max $FF)
        bne.s   FaiTabba
        rts

*****************************************************************************
; ANIMAZIONE: 8 fotogrammi larghi 40*29 pixel, a 8 colori (3 bitplanes)
*****************************************************************************

; Animazione. ogni frame misura 40*29 pixel, 3 bitplanes. Tot. 8 frames

cannoanim:
        incbin  "frame1"        ; 40*29 a 3 bitplanes (8 colori)
        incbin  "frame2"
        incbin  "frame3"
        incbin  "frame4"
        incbin  "frame5"
        incbin  "frame6"
        incbin  "frame7"
        incbin  "frame8"
FineAnim:

*****************************************************************************
;                       COPPERLISTOZZA
*****************************************************************************

        Section Copper,DATA_C

COPPER:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop

        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "

; 8 Colori

        dc.w    $180,$000,$182,$080,$184,$8c6
        dc.w    $186,$c20,$188,$d50,$18a,$e80,$18c,$0fb0
        dc.w    $18e,$ff0

        dc.w    $2c07,$FFFE     ; wait

        dc.w    $100,BPLC0      ; bplcon0 - 3 planes

        dc.w    $108,-40        ; modulo negativo - ripeti stessa linea!
        dc.w    $10A,-40
AllungaCop:
        ds.b    6*4*28          ; 2 wait + 4 move = 6*4 bytes * 21 loops
```

```
                                ; Questa copperlist allunga *8 cio' che e'
                                ; visualizzato, usando i moduli 0 e -40
                                ; alternati ogni 8 linee.
        ds.b    4*2             ; Per il $ffdffffe e per l'ultimo wait

        dc.w    $100,$200       ; bplcon0 - no bitplanes
        dc.w    $FFFF,$FFFE     ; Fine copperlist

*****************************************************************************
; Buffer per la tabella per lo zoom "precalcolato"
*****************************************************************************

        section precalcolone,bss

PrecalcTabba:
        ds.b    256*8

*****************************************************************************
; Buffer dove viene "espanso" ogni fotogramma.
*****************************************************************************

        SECTION BitPlanes,BSS_C

PLANEXPAND:                     ; Dove viene espanso ogni fotogramma.
        ds.b    40*29*3         ; 40 bytes * 29 linee * 3 bitplanes

        end
```

Lezione11l6

```
; Lezione11l6.s         Routine di gestione del modo interlacciato (640x512)
;                       che legge il bit 15 (LOF) del VPOSR ($dff004).
;                       Premendo il tasto destro non si esegue tale routine,
;                       e si nota come rimangano alle volte le linee pari o
;                       le dispari in "pseudo-non lace".

        SECTION Interlace,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30

scr_bytes       = 80    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
```

```
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 2     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 1     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 1     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:

;       Puntiamo i bitplanes in copperlist

        MOVE.L  #BITPLANE,d0    ; in d0 mettiamo l'indirizzo del bitplane
        LEA     BPLPOINTERS,A1  ; puntatori nella COPPERLIST
        move.w  d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap    d0              ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w  d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane


        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$01000,d2      ; linea da aspettare = $010
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $010
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $010
        Beq.S   Waity2

        btst    #2,$16(A5)      ; Tasto destro premuto?
        beq.s   NonLaceint

        bsr.s   laceint         ; Routine che punta linee pari o dispari
                                ; ogni frame a seconda del bit LOF per
                                ; l'interlace
NonLaceint:
        bsr.w   PrintCarattere  ; Stampa un carattere alla volta

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
```

```
        rts

*****************************************************************************
; INTERLACE ROUTINE - Testa il bit LOF (Long Frame) per sapere se si devono
; visualizzare le linee pari o quelle dispari, e punta di conseguenza.
*****************************************************************************

LACEINT:
        MOVE.L  #BITPLANE,D0    ; Indirizzo bitplane
        btst.b  #15-8,4(A5)     ; VPOSR LOF bit?
        Beq.S   Faidispari      ; Se si, tocca alle linee dispari
        ADD.L   #scr_bytes,D0           ; Oppure aggiungi la lunghezza di una linea,
                                ; facendo partire la visualizzazione dalla
                                ; seconda: visualizzate linee pari!
FaiDispari:
        LEA     BPLPOINTERS,A1  ; PLANE POINTERS IN COPLIST
        MOVE.W  D0,6(A1)        ; Punta la figura
        SWAP    D0
        MOVE.W  D0,2(A1)
        RTS

*****************************************************************************
;                       Routine di Print
*****************************************************************************

PRINTcarattere:
        MOVE.L  PuntaTESTO(PC),A0 ; Indirizzo del testo da stampare in a0
        MOVEQ   #0,D2           ; Pulisci d2
        MOVE.B  (A0)+,D2        ; Prossimo carattere in d2
        CMP.B   #$ff,d2         ; Segnale di fine testo? ($FF)
        beq.s   FineTesto       ; Se si, esci senza stampare
        TST.B   d2              ; Segnale di fine riga? ($00)
        bne.s   NonFineRiga     ; Se no, non andare a capo

        ADD.L   #scr_bytes*7,PuntaBITPLANE      ; ANDIAMO A CAPO
        ADDQ.L  #1,PuntaTesto           ; primo carattere riga dopo
                                        ; (saltiamo lo ZERO)
        move.b  (a0)+,d2                ; primo carattere della riga dopo
                                        ; (saltiamo lo ZERO)

NonFineRiga:
        SUB.B   #$20,D2         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        LSL.W   #3,D2           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D2,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        MOVE.L  PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,scr_bytes(A3)     ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,scr_bytes*2(A3)   ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,scr_bytes*3(A3)   ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,scr_bytes*4(A3)   ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,scr_bytes*5(A3)   ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,scr_bytes*6(A3)   ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,scr_bytes*7(A3)   ; stampa LA LINEA 8  " "
```

```
        ADDQ.L  #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.L  #1,PuntaTesto   ; prossimo carattere da stampare

FineTesto:
        RTS

PuntaTesto:
        dc.l    TESTO

PuntaBitplane:
        dc.l    BITPLANE

;       $00 per "fine linea" - $FF per "fine testo"

                ; numero caratteri per linea: 40
TESTO:       ;           1111111111222222222233333333334
             ;  12345678901234567890123456789012345678901
        dc.b    ' Che scritte piccole! Non si leggono nem'   ; 1
        dc.b    'meno... ma sono in 640x512!             ',0 ; 1b
;
        dc.b    'Provate a premere il tasto destro e potr'   ; 2
        dc.b    'ete verificare cosa vedono i coder che  ',0 ; 2b
;
        dc.b    "non sanno come funziona l'interlace, hah"   ; 3
        dc.b    "aha! In fondo e' semplice, no?          ",0 ; 3b
;
        dc.b    'Programmate, fate qualche demo o qualche'   ; 4
        dc.b    " gioco, e' la cosa piu' creativa che si ",0 ; 4b
;
        dc.b    'possa fare nel mondo contemporaneo.     '   ; 5
        dc.b    '                                        ',$FF ; 5b - FINE

        EVEN


;       Il FONT caratteri 8x8.

FONT:
        incbin  "assembler2:sorgenti4/nice.fnt"

*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop

        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,80         ; Bpl1Mod \ INTERLACE: modulo = lungh. linea!
        dc.w    $10a,80         ; Bpl2Mod / per saltarle (le pari o le disp.)

                    ; 5432109876543210
;       dc.w    $100,%1001001000000100  ; 1 bitplan, HIRES LACE 640x512
;                                       ; notare il bit 2 settato per LACE!!

        dc.w    $100,BPLC0      ; BplCon0 -> facciamo calcolare automatico!
```

```
BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $180,$226       ; color0 - SFONDO
        dc.w    $182,$0b0       ; color1 - plane 1 posizione normale, e'
                                ; la parte che "sporge" in alto.

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************

        SECTION MIOPLANE,BSS_C

BITPLANE:
        ds.b    scr_bytes*scr_h ; 80*512 un bitplane Hires int. 640x512

        end

Da notare che e' stato utilizzato il sistema del calcolo automatico dei
diwstart/stop eccetera. Comunque per l'interlace occorre ricordarsi di
mettere il modulo a "scr_bytes", in questo caso 80.
```

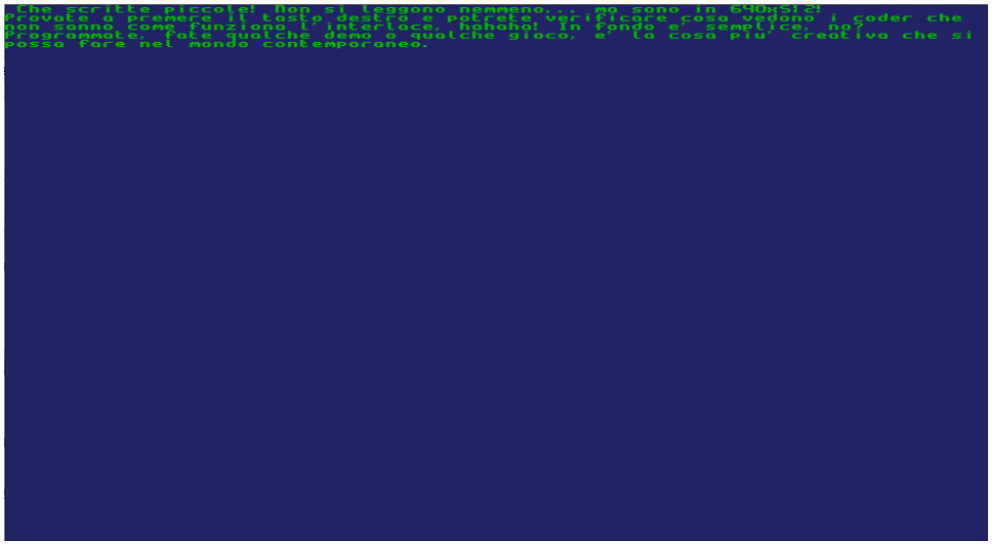


Figura 27.19: Lezione 11l6

Lezione11l6b

```
; Lezione11l6b.s        Routine di gestione del modo interlacciato (640x512)
;                       che legge il bit 15 (LOF) del VPOSR ($dff004).
;                       Premendo il tasto destro non si esegue tale routine,
;                       e si nota come rimangano alle volte le linee pari o
;                       le dispari in "pseudo-non lace".

        SECTION Interlaccione,CODE
```

```
;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; solo copper e bitplane DMA

WaitDisk        EQU     30

scr_bytes       = 40    ; Numero di bytes per ogni linea orizzontale.
                        ; Da questa si calcola la larghezza dello schermo,
                        ; moltiplicando i bytes per 8: schermo norm. 320/8=40
                        ; Es. per uno schermo largo 336 pixel, 336/8=42
                        ; larghezze esempio:
                        ; 264 pixel = 33 / 272 pixel = 34 / 280 pixel = 35
                        ; 360 pixel = 45 / 368 pixel = 46 / 376 pixel = 47
                        ; ... 640 pixel = 80 / 648 pixel = 81 ...

scr_h           = 256   ; Altezza dello schermo in linee
scr_x           = $81   ; Inizio schermo, posizione XX (normale $xx81) (129)
scr_y           = $2c   ; Inizio schermo, posizione YY (normale $2cxx) (44)
scr_res         = 1     ; 2 = HighRes (640*xxx) / 1 = LowRes (320*xxx)
scr_lace        = 1     ; 0 = non interlace (xxx*256) / 1 = interlace (xxx*512)
ham             = 0     ; 0 = non ham / 1 = ham
scr_bpl         = 4     ; Numero Bitplanes

; parametri calcolati automaticamente

scr_w           = scr_bytes*8           ; larghezza dello schermo
scr_size        = scr_bytes*scr_h       ; dimensione in bytes dello schermo
BPLC0   = ((scr_res&2)<<14)+(scr_bpl<<12)+$200+(scr_lace<<2)+(ham<<11)
DIWS    = (scr_y<<8)+scr_x
DIWSt   = ((scr_y+scr_h/(scr_lace+1))&255)<<8+(scr_x+scr_w/scr_res)&255
DDFS    = (scr_x-(16/scr_res+1))/2
DDFSt   = DDFS+(8/scr_res)*(scr_bytes/2-scr_res)


START:
;       puntiamo la figura

        MOVE.L  #Logo1,d0       ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        MOVEQ   #4-1,D1         ; numero di bitplanes (qua sono 4)
POINTBP:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        ADD.L   #40*84,d0       ; + lunghezza bitplane (qua e' alto 84 linee)
        addq.w  #8,a1
        dbra    d1,POINTBP


        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
```

```
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$01000,d2      ; linea da aspettare = $000
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Waity2:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        Beq.S   Waity2

        btst    #2,$16(A5)      ; Tasto destro premuto?
        beq.s   NonLaceint

        bsr.s   laceint         ; Routine che punta linee pari o dispari
                                ; ogni frame a seconda del bit LOF per
                                ; l'interlace
NonLaceint:
        btst.b  #6,$bfe001      ; Mouse sinistro premuto?
        bne.s   mouse
        rts

*****************************************************************************
; INTERLACE ROUTINE - Testa il bit LOF (Long Frame) per sapere se si devono
; visualizzare le linee pari o quelle dispari, e punta di conseguenza.
*****************************************************************************

LACEINT:
        MOVE.L  #Logo1,D0       ; Indirizzo bitplanes
        btst.b  #15-8,4(A5)     ; VPOSR LOF bit?
        Beq.S   Faidispari      ; Se si, tocca alle linee dispari
        ADD.L   #40,D0          ; Oppure aggiungi la lunghezza di una linea,
                                ; facendo partire la visualizzazione dalla
                                ; seconda: visualizzate linee pari!
FaiDispari:
        LEA     BPLPOINTERS,A1  ; PLANE POINTERS IN COPLIST
        MOVEQ   #4-1,D7         ; NUM. DI BITPLANES -1
LACELOOP:
        MOVE.W  D0,6(A1)        ; Punta la figura
        SWAP    D0
        MOVE.W  D0,2(A1)
        SWAP    D0
        ADD.L   #40*84,D0       ; Lunghezza bitplane
        ADDQ.w  #8,A1           ; Prossimi pointers
        DBRA    D7,LACELOOP
        RTS

*****************************************************************************
;                       Copper List
*****************************************************************************
        section copper,data_c           ; Chip data

Copperlist:
        dc.w    $8e,DIWS        ; DiwStrt
        dc.w    $90,DIWSt       ; DiwStop
        dc.w    $92,DDFS        ; DdfStart
        dc.w    $94,DDFSt       ; DdfStop
```

```
        dc.w    $102,0          ; BplCon1 - scroll register
        dc.w    $104,0          ; BplCon2 - priority register
        dc.w    $108,40         ; Bpl1Mod - \ INTERLACE: lungh. di una linea
        dc.w    $10a,40         ; Bpl2Mod - / per saltare linee pari o disp.

; Bitplane pointers

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane
        dc.w $e4,$0000,$e6,$0000        ;secondo bitplane
        dc.w $e8,$0000,$ea,$0000        ;terzo   bitplane
        dc.w $ec,$0000,$ee,$0000        ;quarto  bitplane

;                   ; 5432109876543210
;       dc.w    $100,%0100001000000100  ; BPLCON0 - 4 planes lowres (16 colori)
;                                       ; INTERLACCIATO (bit 2!)

        dc.w    $100,BPLC0      ; BplCon0 - calcolato automaticamente


; i primi 16 colori sono per il LOGO

        dc.w $180,$000,$182,$fff,$184,$200,$186,$310
        dc.w $188,$410,$18a,$620,$18c,$841,$18e,$a73
        dc.w $190,$b95,$192,$db6,$194,$dc7,$196,$111
        dc.w $198,$222,$19a,$334,$19c,$99b,$19e,$446

;       Mettiamo un poco di sfumature per la scenografia...

        dc.w    $5607,$fffe     ; Wait - $2c+84=$80
        dc.w    $100,$204       ; bplcon0 - no bitplanes, MA BIT LACE SETTATO!
        dc.w    $8007,$fffe     ; wait
        dc.w    $180,$003       ; color0
        dc.w    $8207,$fffe     ; wait
        dc.w    $180,$005       ; color0
        dc.w    $8507,$fffe     ; wait
        dc.w    $180,$007       ; color0
        dc.w    $8a07,$fffe     ; wait
        dc.w    $180,$009       ; color0
        dc.w    $9207,$fffe     ; wait
        dc.w    $180,$00b       ; color0

        dc.w    $9e07,$fffe     ; wait
        dc.w    $180,$999       ; color0
        dc.w    $a007,$fffe     ; wait
        dc.w    $180,$666       ; color0
        dc.w    $a207,$fffe     ; wait
        dc.w    $180,$222       ; color0
        dc.w    $a407,$fffe     ; wait
        dc.w    $180,$001       ; color0

        dc.l    $ffff,$fffe     ; Fine della copperlist


*****************************************************************************
;                               DISEGNO
*****************************************************************************

        section gfxstuff,data_c

; Disegno largo 320 pixel, alto 84, a 4 bitplanes (16 colori).
```

```
Logo1:
        incbin  'assembler2:sorgenti4/logo320*84*16c.raw'

        end
```

Avete notato che TUTTI i bplcon0 della copperlist devono avere il bit 2, quello dell'interlace, settato? Infatti, se l'ultimo BPLCON0 non avesse il bit settato, nonostante gli altri lo abbiano lo schermo non verrebbe interlacciato!

Lezione11l7

```
; Lezione11l7.s         8 sprites attacched (quindi 4 a 16 colori) usati,
;                       anzi "riutilizzati" 128 volte per linea.

        SECTION MegaRiuso,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper,bitplane,sprites
;                -----a-bcdefghij

Waitdisk        EQU     30

NumeroLinee     =       128
LungSpr         =       NumeroLinee*8

START:

; Punta gli sprites

        MOVE.L  #SpritesBuffer,d0
        LEA     SPRITEPOINTERS,A1
        MOVEQ   #8-1,D1                 ;num of sprites = 8
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #LungSpr,d0             ;lenght of sprite
        addq.w  #8,a1
        dbra    d1,POINTB               ;Rifai D1 volte

; Puntiamo il biplane azzerato

        MOVE.L  #PLANE,d0
        LEA     BPLPOINTERS,A1
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        bsr.s   CreaSprites     ; routine che crea i 4 sprite attacched,
                                ; ossia tutti e 8 gli sprite, fatti da
                                ; 128 riutilizzi di 1 linea ognuno!
```

```
        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  #COPPER,$80(a5)         ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12c00,d2      ; linea da aspettare = $12c
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $010
        BNE.S   Waity1

        btst    #2,$16(A5)      ; Tasto destro premuto?
        beq.s   NonOndegg

        bsr.w   OndeggiaSpriteS ; ondeggia gli 8 sprites riutilizzati

NonOndegg:
        btst    #6,$bfe001      ; tasti sin. mouse premuto?
        bne.s   mouse
        rts

; ***************************************************************************
; Routine che crea gli 8 sprites, (ossia 4 attacched) nello "SpritesBuffer".
; Da notare che gli sprite attacched sono a sua volta affiancati a 2 a 2,
; in modo da ottenere 2 barre larghe 16*2=32 pixel, a 16 colori.
; Innanzitutto occorre ricordare che ogni sprite si puo' "riutilizzare",
; ossia "sotto" ad uno sprite, dopo la fine dello sprite, si puo'
; mettere un'altro sprite, a patto pero' che la sua posizione verticale
; di inizio lasci 1 linea "vuota". Qua usiamo questo fatto in modo massiccio,
; infatti ogni riuso dello sprite e' di 1 linea, per cui otteniamo una
; striscia verticale (larga 16 pixel) fatta di tanti "spritini" alti una linea
; distanziati da una linea "vuota". Per riempire le 256 linee verticali dello
; schermo, facciamo ben 128 utilizzi per ogni sprite! Ma almeno possiamo
; "curvare" quella linea di quanto vogliamo, dato che ogni striscia ha un
; proprio HSTART (posizione orizzontale) indipendente.
;
; Ricordiamo la struttura di uno sprite:
;
;VSTART:
;       dc.b xx         ; Pos. verticale (da $2c a $f2)
;HSTART:
;       dc.b xx+(xx)    ; Pos. orizzontale (da $40 a $d8)
;VSTOP:
;       dc.b xx         ; fine verticale.
;       dc.b $00        ; byte speciale: bit 7 per ATTACCHED!!
;       dc.l    XXXXX   ; bitplane dello sprite (disegno!) qua 1 linea
;       dc.w    0,0     ; 2 word azzerate di FINE SPRITE, che qua mettiamo
;                       ; mai... quindi qua ci saranno gia' i VSTART e il
;                       ; VSTOP del prossimo sprite!
;
; 4 bytes -> words di controllo + 4 bytes -> figura (1 striscia)
; 4*2= 8 -> lunghezza di uno sprite; 8*128 = 1024, lunghezza 1 sprite.
; facciamo 128 riutilizzi di ogni sprite: 2 linee per sprite = 256 linee!
```

```
;
; ***************************************************************************

; 1024 bytes (8*128) a sprite

CreaSprites:
        lea     SpritesBuffer,A0 ; destinazione
        move.l  #%10000000,D5   ; bit 7 settato - per attacched in sprite+3
        moveq   #$2c,D0         ; VSTART - iniziamo da $2c
CreaLoop:
        move.b  d0,(A0)         ; metti il vstart agli 8 sprite
        move.b  d0,LungSpr(A0)  ; 2 (ogni sprite e' lungo 2400 bytes)
        move.b  d0,LungSpr*2(A0)        ; 3
        move.b  d0,LungSpr*3(A0)        ; 4
        move.b  d0,LungSpr*4(A0)        ; 5
        move.b  d0,LungSpr*5(A0)        ; 6
        move.b  d0,LungSpr*6(A0)        ; 7
        move.b  d0,LungSpr*7(A0)        ; 8

        move.l  d0,D1
        addq.w  #1,D1           ; VSTART 1 linea sotto -> usiamolo come VSTOP

        move.b  d1,2(A0)                ; metti il vstop agli 8 sprite
        move.b  d1,LungSpr+2(A0)        ; 2 (ogni sprite e' lungo 2400 bytes)
        move.b  d1,(LungSpr*2)+2(A0)    ; 3
        move.b  d1,(LungSpr*3)+2(A0)    ; 4
        move.b  d1,(LungSpr*4)+2(A0)    ; 5
        move.b  d1,(LungSpr*5)+2(A0)    ; 6
        move.b  d1,(LungSpr*6)+2(A0)    ; 7
        move.b  d1,(LungSpr*7)+2(A0)    ; 8

; Settiamo i bit attacched agli 8 sprite

        move.b  d5,3(A0)                ; metti il byte spec. agli 8 sprite
        move.b  d5,LungSpr+3(A0)        ; 2 (ogni sprite e' lungo 2400 bytes)
        move.b  d5,(LungSpr*2)+3(A0)    ; 3
        move.b  d5,(LungSpr*3)+3(A0)    ; 4
        move.b  d5,(LungSpr*4)+3(A0)    ; 5
        move.b  d5,(LungSpr*5)+3(A0)    ; 6
        move.b  d5,(LungSpr*6)+3(A0)    ; 7
        move.b  d5,(LungSpr*7)+3(A0)    ; 8

        addq.w  #4,A0                   ; saltiamo le 2 word di controllo
                                        ; e andiamo nei plane degli sprite!

        move.l  #$55553333,(A0)         ; 1 \ metti la linea sfumata
        move.l  #$0f0f00ff,LungSpr(A0)  ; 2 / attacched 1!

        move.l  #$aaaacccc,LungSpr*2(A0)        ; 3 \ attacched 2!
        move.l  #$f0f0ff00,LungSpr*3(A0)        ; 4 /

        move.l  #$55553333,LungSpr*4(A0)        ; 5 \ attacched 3!
        move.l  #$0f0f00ff,LungSpr*5(A0)        ; 6 /

        move.l  #$aaaacccc,LungSpr*6(A0)        ; 7 \ attacched 4!
        move.l  #$f0f0ff00,LungSpr*7(A0)        ; 8 /

        addq.w  #4,A0                   ; saltiamo le 2 word dei plane,
                                        ; per andare alle prossime
                                        ; 2 word di controllo, dato che
                                        ; non ci sono le 2 word azzerate
                                        ; di fine sprite.
```

```
        cmp.b   #%10000110,D5   ; siamo sotto la linea $FF?
        beq.s   SiamoSottoFF
        addq.b  #2,D0           ; vstart 2 linee sotto per il prossimo
                                ; riutilizzo dello sprite. Dato che ogni
                                ; sprite e' alto 1 linea, e che tra un
                                ; utilizzo ed un altro occorre lasciare
                                ; una linea vuota, addiamo 2.
        bne.w   CreaLoop        ; siamo giunti a $fe+2 = $00?
                                ; Se si, occorre settare il bit alto di
                                ; vstart e vstop. Altrimenti continua

        move.b  #%10000110,D5   ; %10000110 -> settati i 2 bit alti di vstart
                                ; e vstop per andare sotto la linea $FF
        subq.b  #2,D0           ; ritorniamo "indietro" di 1 passo...

SiamoSottoFF:
        addq.b  #2,D0           ; vstart 2 linee sotto...
        cmpi.b  #$2c,D0         ; siamo alla posizione $FF+$2c?
        bne.w   CreaLoop        ; se non ancora, continua!
        rts

; ***************************************************************************

; Parametri per "IS"

; BEG> 0
; END> 360
; AMOUNT> 250
; AMPLITUDE> $20
; YOFFSET> $20
; SIZE (B/W/L)> b
; MULTIPLIER> 1

SinTabHstarts:
 dc.B   $20,$21,$22,$23,$24,$24,$25,$26,$27,$28,$28,$29,$2A,$2B,$2B,$2C
 dc.B   $2D,$2E,$2E,$2F,$30,$30,$31,$32,$32,$33,$34,$34,$35,$36,$36,$37
 dc.B   $37,$38,$38,$39,$39,$3A,$3A,$3B,$3B,$3C,$3C,$3C,$3D,$3D,$3D,$3E
 dc.B   $3E,$3E,$3F,$3F,$3F,$3F,$3F,$40,$40,$40,$40,$40,$40,$40,$40,$40
 dc.B   $40,$40,$40,$40,$40,$40,$3F,$3F,$3F,$3F,$3F,$3E,$3E,$3E,$3D,$3D
 dc.B   $3D,$3C,$3C,$3C,$3B,$3B,$3A,$3A,$39,$39,$38,$38,$37,$37,$36,$36
 dc.B   $35,$34,$34,$33,$32,$32,$31,$30,$30,$2F,$2E,$2E,$2D,$2C,$2B,$2B
 dc.B   $2A,$29,$28,$28,$27,$26,$25,$24,$24,$23,$22,$21,$20,$20,$1F,$1E
 dc.B   $1D,$1C,$1C,$1B,$1A,$19,$18,$18,$17,$16,$15,$15,$14,$13,$12,$12
 dc.B   $11,$10,$10,$0F,$0E,$0E,$0D,$0C,$0C,$0B,$0A,$0A,$09,$09,$08,$08
 dc.B   $07,$07,$06,$06,$05,$05,$04,$04,$04,$03,$03,$03,$02,$02,$02,$01
 dc.B   $01,$01,$01,$01,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
 dc.B   $00,$00,$00,$01,$01,$01,$01,$01,$02,$02,$02,$03,$03,$03,$04,$04
 dc.B   $04,$05,$05,$06,$06,$07,$07,$08,$08,$09,$09,$0A,$0A,$0B,$0C,$0C
 dc.B   $0D,$0E,$0E,$0F,$10,$10,$11,$12,$12,$13,$14,$15,$15,$16,$17,$18
 dc.B   $18,$19,$1A,$1B,$1C,$1C,$1D,$1E,$1F,$20
FinTab:

TabLunghezz     = FinTab-SinTabHstarts

OndeggiaSpriteS:
        addq.b  #1,Barra1OffSalv
        moveq   #0,D0
        move.b  Barra1OffSalv(pc),D0
        cmp.w   #TabLunghezz,D0         ; siamo al massimo offset?
        bne.s   NonRipartireO1
        clr.b   Barra1OffSalv           ; riparti da capo
```

```
NonRipartire01:
        addq.b  #2,Barra2OffSalv
        moveq   #0,D0
        move.b  Barra2OffSalv(pc),D0
        cmp.w   #TabLunghezz,D0
        bne.s   NonRipartire02
        clr.b   Barra2OffSalv           ; riparti da capo
NonRipartire02:
        moveq   #0,D1
        moveq   #0,D2
        moveq   #0,D3
        moveq   #0,D4
        moveq   #0,D5
        lea     SpritesBuffer,A0        ; indirizzo primo sprite
        lea     SinTabHstarts(PC),A1
        move.b  Barra1OffSalv(pc),D0
        move.b  Barra2OffSalv(pc),D2
        move.b  0(A1,D0.w),D5   ; da sintab secondo Barra1OffSalv
OndeggiaLoop:
        move.b  0(A1,D0.w),D3   ; da sintab - per barra 1
        move.b  0(A1,D2.w),D4   ; da sintab - per barra 2

; modifica tutto

        add.b   D4,D3   ; barra 1
        sub.b   D5,D3   ;

        add.b   D5,D4   ; barra 2

        add.b   #105,D3 ; centra barra 1
        add.b   #75,D4  ; centra barra 2

; Modifica gli HSTART (posizione orizzontale) degli 8 sprites

; ** Prima Barra

        move.b  D3,1(A0)                ; sprite 1
        move.b  D3,LungSpr+1(A0)        ; 2

; ora lo sprite attacched della stessa barra, ma affiancato (16 pixel dopo)

        addq.w  #8,D3                   ; adda 8, ossia 16 pixel, dato che
                                        ; HSTART adda 2 ogni volta.
        move.b  D3,(LungSpr*2)+1(A0)    ; 3
        move.b  D3,(LungSpr*3)+1(A0)    ; 4

; ** Seconda Barra

        move.b  D4,(LungSpr*4)+1(A0)    ; 5
        move.b  D4,(LungSpr*5)+1(A0)    ; 6

        addq.w  #8,D4                   ; adda 8, ossia 16 pixel, dato che
                                        ; HSTART adda 2 ogni volta.
        move.b  D4,(LungSpr*6)+1(A0)    ; 7
        move.b  D4,(LungSpr*7)+1(A0)    ; 8

        addq.w  #1,D2           ; offset prossimo - bar 2...
        cmpi.w  #TabLunghezz,D2 ; siamo al massimo?
        bne.s   Nonrestart2
        moveq   #0,D2           ; rileggi dal primo valore...
Nonrestart2:
        addq.w  #1,D0           ; offset prossimo - bar 1
```

```
        cmp.w   #TabLunghezz,D0 ; siamo al massimo?
        bne.s   Nonrestart1
        moveq   #0,D0           ; rileggi dal primo valore
Nonrestart1:
        addq.w  #8,A0           ; salta al prossimo riutilizzo di sprite

        cmpa.l  #SpritesBuffer+LungSpr,a0 ; abbiamo finito?
        bne.s   OndeggiaLoop
        rts

Barra1OffSalv:
        dc.w    0
Barra2OffSalv:
        dc.w    0


; ***************************************************************************
;                               COPPERLIST
; ***************************************************************************

        section baucoppe,data_c

COPPER:
        dc.w    $8e,$2c81       ; diwstart
        dc.w    $90,$2cc1       ; diwstop
        dc.w    $92,$38         ; ddfstart
        dc.w    $94,$d0         ; ddfstop

SPRITEPOINTERS:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0,$12a,0,$12c,0,$12e,0
        dc.w    $130,0,$132,0,$134,0,$136,0,$138,0,$13a,0,$13c,0,$13e,0

        dc.w    $108,0  ; bpl1mod
        dc.w    $10a,0  ; bpl2mod
        dc.w    $102,0  ; bplcon1
        dc.w    $104,0  ; bplcon2

BPLPOINTERS:
        dc.w    $e0,0,$e2,0     ; plane 1

        dc.w    $100,$1200      ; bplcon0 - 1 plane lowres

        dc.w    $180,0          ; color0 - nero
        dc.w    $182,$fff       ; color1 - bianco

; Colori degli sprite (attacched) - da color17 a color31

        dc.w    $1a2,$010,$1a4,$020,$1a6,$030
        dc.w    $1a8,$140,$1aa,$250,$1ac,$360,$1ae,$470
        dc.w    $1b0,$580,$1b2,$690,$1b4,$7a0,$1b6,$8b0
        dc.w    $1b8,$9c0,$1ba,$ad0,$1bc,$be0,$1be,$cf0

        dc.w    $ffff,$fffe     ; fine copperlist

; ***************************************************************************

        section grafica,bss_C

SpritesBuffer:
        DS.B    LungSpr*8       ; 1024 bytes ogni sprite megariutilizzato

; ***************************************************************************
```

```
plane:
        ds.b    40*256  ; 1 plane lowres "nero" come sfondo.

        END
```

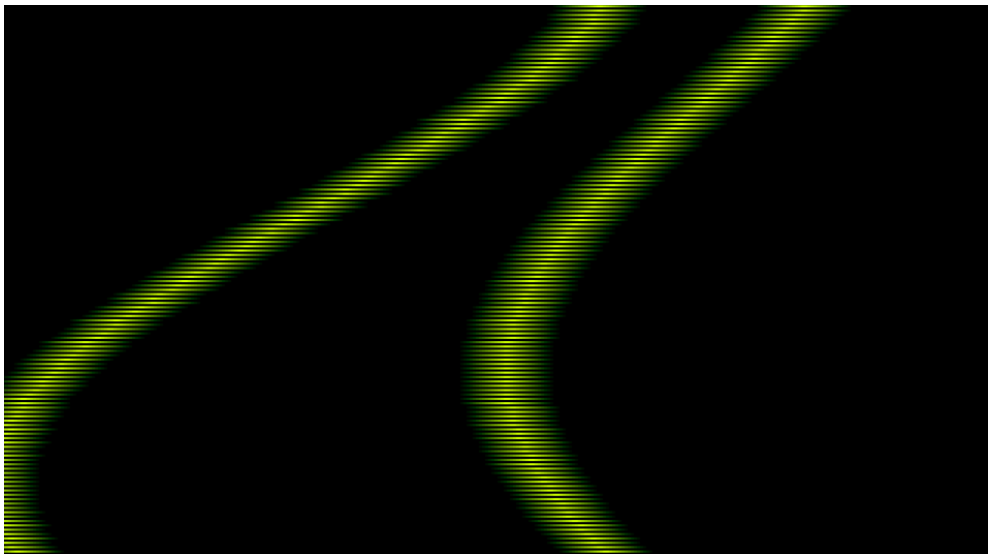


Figura 27.20: Lezione 11l7

27.11 Lezione11m1

```
; Lezione11m1.s - Utilizzo dell'interrupt di livello 2 ($68) per leggere i
;                 codici dei tasti premuti sulla tastiera.
;                 PREMERE LO SPAZIO PER USCIRE

        Section InterruptKey,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
```

```
        MOVE.L  #MioInt68KeyB,$68(A0)   ; Routine per la tastiera int. liv. 2
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3

                ; 76543210
        move.b  #%01111111,$bfed01      ; CIAAICR - Disabilita tutte le CIA IRQ
        move.b  #%10001000,$bfed01      ; CIAAICR - Attiva solo la SP CIA IRQ

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

                ; 5432109876543210
        move.w  #%1100000000101000,$9a(a5)    ; INTENA - abilito solo VERTB
                                              ; del livello 3 e il livello2

AttendiSpazio:
        move.b  ActualKey(PC),d0 ; Prendi il codice dell'ultimo tasto premuto.
        move.b  d0,Color0+1     ; Metti il cod. del carattere attuale come
                                ; color0... giusto per test..
        cmp.b   #$40,d0         ; BARRA SPAZIO PREMUTA? (basta col mouse!)
        bne.s   AttendiSpazio

        bsr.w   mt_end              ; fine del replay!
        move.b  #%10011111,$bfed01 ; CIAAICR - Riabilita tutte le CIA IRQ
        rts                         ; esci

; Variabile dove e' salvato il carattere attuale

ActualKey:
        dc.b    0

        even

*****************************************************************************
*       ROUTINE IN INTERRUPT $68 (livello 2) - gestione TASTIERA
*****************************************************************************

;03     PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2

MioInt68KeyB:   ; $68
        movem.l d0/a0,-(sp)     ; salva i registri usati nello stack
        lea     $dff000,a0      ; reg. custom per offset

        MOVE.B  $BFED01,D0      ; Ciaa icr - in d0 (leggendo l'icr causiamo
                                ; anche il suo azzeramento, per cui l'int e'
                                ; "disdetto" come in intreq).
        BTST.l  #7,D0   ; bit IR, (interrupt cia autorizzato), azzerato?
        BEQ.s   NonKey  ; se si, esci
        BTST.l  #3,D0   ; bit SP, (interrupt della tastiera), azzerato?
        BEQ.s   NonKey  ; se si, esci

        MOVE.W  $1C(A0),D0      ; INTENAR in d0
        BTST.l  #14,D0          ; Bit Master di abilitazione azzerato?
        BEQ.s   NonKey          ; Se si, interrupt non attivi!
```

```
        AND.W   $1E(A0),D0      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        btst.l  #3,d0           ; INTREQR - PORTS?
        beq.w   NonKey          ; Se no, allora esci!

; Dopo i controlli, se siamo qua significa che dobbiamo prendere il carattere!

        moveq   #0,d0
        move.b  $bfec01,d0      ; CIAA sdr (serial data register - connesso
                                ; alla tastiera - contiene il byte inviato dal
                                ; chip della tastiera) LEGGIAMO IL CHAR!

; abbiamo il char in d0, lo "lavoriamo"...

        NOT.B   D0              ; aggiustiamo il valore invertendo i bit
        ROR.B   #1,D0           ; e riportando la sequenza a 76543210.
        move.b  d0,ActualKey    ; salviamo il carattere

; Ora dobbiamo comunicare alla tastiera che abbiamo preso il dato!

        bset.b  #6,$bfee01      ; CIAA cra - sp ($bfec01) output, in modo da
                                ; abbassare la linea KDAT per confermare che
                                ; abbiamo ricevuto il carattere.

        st.b    $bfec01         ; $FF in $bfec01 - ue'! ho ricevuto il dato!

; Qua dobbiamo mettere una routine che aspetti 90 microsecondi perche' la
; linea KDAT deve stare bassa abbastanza tempo per essere "capita" da tutti
; i tipi di tastiere. Si possono, per esempio, aspettare 3 o 4 linee raster.

        moveq   #4-1,d0 ; Numero di linee da aspettare = 4 (in pratica 3 piu'
                        ; la frazione in cui siamo nel momento di inizio)
waitlines:
        move.b  6(a0),d1        ; $dff006 - linea verticale attuale in d1
stepline:
        cmp.b   6(a0),d1        ; siamo sempre alla stessa linea?
        beq.s   stepline        ; se si aspetta
        dbra    d0,waitlines    ; linea "aspettata", aspetta d0-1 linee

; Ora che abbiamo atteso, possiamo riportare $bfec01 in modo input...

        bclr.b  #6,$bfee01      ; CIAA cra - sp (bfec01) input nuovamente.

NonKey:         ; 3210
        move.w  #%1000,$9c(a0)  ; INTREQ togli richiesta, int eseguito!
        movem.l (sp)+,d0/a0     ; ripristina i registri dallo stack
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.      *
*****************************************************************************

;06     BLIT    3 ($6c) Se il blitter ha finito una blittata si setta ad 1
;05     VERTB   3 ($6c) Generato ogni volta che il pennello elettronico e'
;                       alla linea 00, ossia ad ogni inizio di vertical blank.
;04     COPER   3 ($6c) Si puo' settare col copper per generarlo ad una certa
;                       linea video. Basta richiederlo dopo un certo WAIT.

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
```

```
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:
NointCOPER:
NoBLIT:          ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,VERTB e COPER
        rte     ; uscita dall'int COPER/BLIT/VERTB

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180
Color0:
        dc.w    $000            ; color0 - sara' cambiato a seconda del tasto
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.fairlight"

        end
```

Immettendo il byte del codice di tastiera nel color0 si nota bene il fatto che quando un tasto e' PREMUTO il bit 7 e' azzerato, mentre quando e' RILASCIATO tale bit e' settato: infatti quando si preme un tasto il colore e' piu' scuro di quando si rilascia, avendo il bit alto del verde azzerato.

Lezione11m2

```
; Lezione11m2.s - Utilizzo dell'interrupt di livello 2 ($68) per leggere i
;                 codici dei tasti premuti sulla tastiera.
;                 In questo caso decodifichiamo anche il tasto letto
;                 trasformandolo nel carattere ASCII corrispondente.

        Section InterruptKey,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************


; Con DMASET decidiamo quali canali DMA aprire e quali chiudere
```

```
                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper,bitplane e DMA abilitato

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo i bitplanes in copperlist

        MOVE.L  #BITPLANE,d0    ; in d0 mettiamo l'indirizzo del bitplane
        LEA     BPLPOINTERS,A1  ; puntatori nella COPPERLIST
        move.w  d0,6(a1)        ; copia la word BASSA dell'indirizzo del plane
        swap    d0              ; scambia le 2 word di d0 (es: 1234 > 3412)
        move.w  d0,2(a1)        ; copia la word ALTA dell'indirizzo del plane

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR

        MOVE.L  #MioInt68KeyB,$68(A0)   ; Routine per la tastiera int. liv. 2
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

                ; 5432109876543210
        move.w  #%1100000000101000,$9a(a5)    ; INTENA - abilito solo VERTB
                                              ; del livello 3 e il livello2
Mouse:
        btst    #6,$bfe001
        bne.s   mouse

        bsr.w   mt_end              ; fine del replay!
        rts                         ; esci

        even

*****************************************************************************
*       ROUTINE IN INTERRUPT $68 (livello 2) - gestione TASTIERA
*****************************************************************************

;03     PORTS   2 ($68) Input/Output Porte e timers, connesso alla linea INT2

MioInt68KeyB:   ; $68
        movem.l d0/a0,-(sp)     ; salva i registri usati nello stack
        lea     $dff000,a0      ; reg. custom per offset

        MOVE.B  $BFED01,D0      ; Ciaa icr - in d0 (leggendo l'icr causiamo
                                ; anche il suo azzeramento, per cui l'int e'
                                ; "disdetto" come in intreq).
        BTST.l  #7,D0   ; bit IR, (interrupt cia autorizzato), azzerato?
        BEQ.s   NonKey  ; se si, esci
        BTST.l  #3,D0   ; bit SP, (interrupt della tastiera), azzerato?
        BEQ.s   NonKey  ; se si, esci
```

```
        MOVE.W  $1C(A0),D0      ; INTENAR in d0
        BTST.l  #14,D0          ; Bit Master di abilitazione azzerato?
        BEQ.s   NonKey          ; Se si, interrupt non attivi!
        AND.W   $1E(A0),D0      ; INREQR - in d1 rimangono settati solo i bit
                                ; che sono settati sia in INTENA che in INTREQ
                                ; in modo da essere sicuri che l'interrupt
                                ; avvenuto fosse abilitato.
        btst.l  #3,d0           ; INTREQR - PORTS?
        beq.w   NonKey          ; Se no, allora esci!

; Dopo i controlli, se siamo qua significa che dobbiamo prendere il carattere!

        moveq   #0,d0
        move.b  $bfec01,d0      ; CIAA sdr (serial data register - connesso
                                ; alla tastiera - contiene il byte inviato dal
                                ; chip della tastiera) LEGGIAMO IL CHAR!

        bsr.s   convertichar    ; Converti in ASCII il carattere

; Ora dobbiamo comunicare alla tastiera che abbiamo preso il dato!

        bset.b  #6,$bfee01      ; CIAA cra - sp ($bfec01) output, in modo da
                                ; abbassare la linea KDAT per confermare che
                                ; abbiamo ricevuto il carattere.

        st.b    $bfec01         ; $FF in $bfec01 - ue'! ho ricevuto il dato!

; Qua dobbiamo mettere una routine che aspetti 90 microsecondi perche' la
; linea KDAT deve stare bassa abbastanza tempo per essere "capita" da tutti
; i tipi di tastiere. Si possono, per esempio, aspettare 3 o 4 linee raster.

        moveq   #4-1,d0 ; Numero di linee da aspettare = 4 (in pratica 3 piu'
                        ; la frazione in cui siamo nel momento di inizio)
waitlines:
        move.b  6(a0),d1        ; $dff006 - linea verticale attuale in d1
stepline:
        cmp.b   6(a0),d1        ; siamo sempre alla stessa linea?
        beq.s   stepline        ; se si aspetta
        dbra    d0,waitlines    ; linea "aspettata", aspetta d0-1 linee

; Ora che abbiamo atteso, possiamo riportare $bfec01 in modo input...

        bclr.b  #6,$bfee01      ; CIAA cra - sp (bfec01) input nuovamente.

NonKey:         ; 3210
        move.w  #%1000,$9c(a0)  ; INTREQ togli richiesta, int eseguito!
        movem.l (sp)+,d0/a0     ; ripristina i registri dallo stack
        rte

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.      *
*****************************************************************************

;06     BLIT    3 ($6c) Se il blitter ha finito una blittata si setta ad 1
;05     VERTB   3 ($6c) Generato ogni volta che il pennello elettronico e'
;                       alla linea 00, ossia ad ogni inizio di vertical blank.
;04     COPER   3 ($6c) Si puo' settare col copper per generarlo ad una certa
;                       linea video. Basta richiederlo dopo un certo WAIT.

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
```

```
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.s   PrintaChar              ; Stampa il carattere
        bsr.w   mt_music                ; suono la musica
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
nointVERTB:
NointCOPER:
NoBLIT:          ;6543210
        move.w  #%1110000,$dff09c ; INTREQ - cancello rich. BLIT,VERTB e COPER
        rte     ; uscita dall'int COPER/BLIT/VERTB


*****************************************************************************
; SubRoutine che converte il carattere in ASCII
*****************************************************************************

ConvertiChar:
        movem.l d1-d2/a0,-(SP)

; data received bit : 6 5 4 3 2 1 0 7
; il bit 7 e' 1 se il tasto e' rilasciato

        not.b   d0                      ;e' trasmesso nottatto
        lsr.b   #1,d0                   ;e ruotato a sx
        bcs.b   Tasto_up

        cmp.b   #$60,d0                 ;left shift
        blo.b   To_Ascii
        bset    d0,Control_Key
        bra.b   exit

Tasto_up:
        cmp.b   #$60,d0                 ;left shift
        blo.b   exit
        bclr    d0,Control_Key
        bra.b   exit

;       bit     7 6     5 4     3    2     1 0
;              Amiga    Alt   Ctrl  caps  shift
;               r l     r l         lock   r l

to_ascii:
        move.b  Control_Key(PC),d1
        beq.b   Get_Char
        move.b  d1,d2
        and.b   #%00000111,d1
        beq.b   tst_alt
        add.w   #$68,d0
        bra.b   Get_Char
tst_alt:
        and.b   #%00110000,d2
;       beq     ....
        add.w   #$68*2,d0
Get_Char:
        lea     Raw_2_Ascii(pc),a0
        move.b  (a0,d0.w),d0
        move.b  d0,ascii_char
        clr.b   received        ; il dato e' pronto!
exit:
        movem.l (SP)+,d1-d2/a0
        rts

*****************************************************************************
```

```
PrintaChar:
        tst.b   received        ; Dato ricevuto?
        bne.s   NonPremuto
        st.b    received
        moveq   #0,d0
        move.b  ascii_char(pc),d0
        cmp.b   #-1,d0
        beq.b   NonValido  ; era un carattere speciale tipo help tab ecc.
        bsr.s   PrintaD0   ; altrimenti stampa il char sullo schermo
NonValido:
NonPremuto:
        rts

Control_Key:    dc.b    0
ascii_char:     dc.b    0
received:       dc.b    -1
contariga:      dc.b    0

        even

*****************************************************************************
; Routine di Print del carattere in d0
*****************************************************************************

PRINTAd0:
        movem.l a2-a3,-(SP)

        SUB.B   #$20,D0         ; TOGLI 32 AL VALORE ASCII DEL CARATTERE, IN
                                ; MODO DA TRASFORMARE, AD ESEMPIO, QUELLO
                                ; DELLO SPAZIO (che e' $20), in $00, quello
                                ; DELL'ASTERISCO ($21), in $01...
        LSL.W   #3,D0           ; MOLTIPLICA PER 8 IL NUMERO PRECEDENTE,
                                ; essendo i caratteri alti 8 pixel
        MOVE.L  D0,A2
        ADD.L   #FONT,A2        ; TROVA IL CARATTERE DESIDERATO NEL FONT...

        cmp.b   #80,ContaRiga   ; 80 caratteri stampati?
        bne.s   NonFine
        add.l   #80*7,PuntaBitplane     ; Vai a Capo
        clr.b   ContaRiga
NonFine:
        MOVE.L  PuntaBITPLANE(PC),A3 ; Indir. del bitplane destinazione in a3

                                ; STAMPIAMO IL CARATTERE LINEA PER LINEA
        MOVE.B  (A2)+,(A3)      ; stampa LA LINEA 1 del carattere
        MOVE.B  (A2)+,80(A3)    ; stampa LA LINEA 2  " "
        MOVE.B  (A2)+,80*2(A3)  ; stampa LA LINEA 3  " "
        MOVE.B  (A2)+,80*3(A3)  ; stampa LA LINEA 4  " "
        MOVE.B  (A2)+,80*4(A3)  ; stampa LA LINEA 5  " "
        MOVE.B  (A2)+,80*5(A3)  ; stampa LA LINEA 6  " "
        MOVE.B  (A2)+,80*6(A3)  ; stampa LA LINEA 7  " "
        MOVE.B  (A2)+,80*7(A3)  ; stampa LA LINEA 8  " "

        ADDQ.L  #1,PuntaBitplane ; avanziamo di 8 bit (PROSSIMO CARATTERE)
        ADDQ.B  #1,ContaRiga

        movem.l (SP)+,a2-a3
        RTS

PuntaBitplane:
        dc.l    BITPLANE
```

```
; Tabella di conversione in ascii. Facilmente modificabile per tastiera
; italiana o altre.

raw_2_ascii:
        dc.b    ’‘’
        dc.b    ’1’
        dc.b    ’2’
        dc.b    ’3’
        dc.b    ’4’
        dc.b    ’5’
        dc.b    ’6’
        dc.b    ’7’
        dc.b    ’8’
        dc.b    ’9’
        dc.b    ’0’
        dc.b    ’-’
        dc.b    ’=’
        dc.b    ’\’
        dc.b    -1      ;<<<<<<<<<<<<<<
        dc.b    ’0’     ;tastierino numerico
        dc.b    ’q’
        dc.b    ’w’
        dc.b    ’e’
        dc.b    ’r’
        dc.b    ’t’
        dc.b    ’y’
        dc.b    ’u’
        dc.b    ’i’
        dc.b    ’o’
        dc.b    ’p’
        dc.b    ’[’
        dc.b    ’]’
        dc.b    -1      ;<<<<<<<<<<<<<<<<<<
        dc.b    ’1’
        dc.b    ’2’
        dc.b    ’3’
        dc.b    ’a’
        dc.b    ’s’
        dc.b    ’d’
        dc.b    ’f’
        dc.b    ’g’
        dc.b    ’h’
        dc.b    ’j’
        dc.b    ’k’
        dc.b    ’l’
        dc.b    ’;’
        dc.b    39
        dc.b    -1      ;not used
        dc.b    -1      ;<<<<<<<<<<<<<<<<<<<<<<
        dc.b    ’4’
        dc.b    ’5’
        dc.b    ’6’
        dc.b    ’<’
        dc.b    ’z’
        dc.b    ’x’
        dc.b    ’c’
        dc.b    ’v’
        dc.b    ’b’
        dc.b    ’n’
        dc.b    ’m’
```

```
dc.b    ','
dc.b    '.'
dc.b    '/'
dc.b    -1      ;<<<<<<<<<<<<<<<<<<
dc.b    '.'
dc.b    '7'
dc.b    '8'
dc.b    '9'
dc.b    ' '     ;space
dc.b    -1      ;back space
dc.b    -1      ;tab
dc.b    -1      ;return tastierino
dc.b    -1      ;return
dc.b    -1      ;esc
dc.b    -1      ;del
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;<<<<<<<<<
dc.b    '-'
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;up
dc.b    -1      ;down
dc.b    -1      ;dx
dc.b    -1      ;sx
dc.b    -1      ;f1
dc.b    -1      ;f2
dc.b    -1      ;f3
dc.b    -1      ;f4
dc.b    -1      ;f5
dc.b    -1      ;f6
dc.b    -1      ;f7
dc.b    -1      ;f8
dc.b    -1      ;f9
dc.b    -1      ;f10
dc.b    '('
dc.b    ')'
dc.b    '/'
dc.b    '*'
dc.b    '+'
dc.b    -1      ;help
dc.b    -1      ;lshift
dc.b    -1      ;rshift
dc.b    -1      ;caps lock
dc.b    -1      ;ctrl
dc.b    -1      ;lalt
dc.b    -1      ;ralt
dc.b    -1      ;lamiga
dc.b    -1      ;ramiga

dc.b    '~'     ;shift-tati
dc.b    '!'
dc.b    '@'
dc.b    '#'
dc.b    '$'
dc.b    '%'
dc.b    '^'
dc.b    '&'
dc.b    '*'
dc.b    '('
dc.b    ')'
dc.b    '_'
dc.b    '+'
```

```
dc.b    '|'
dc.b    -1      ;<<<<<<<<<<<<<<
dc.b    '0'     ;tastierino numerico
dc.b    'Q'
dc.b    'W'
dc.b    'E'
dc.b    'R'
dc.b    'T'
dc.b    'Y'
dc.b    'U'
dc.b    'I'
dc.b    'O'
dc.b    'P'
dc.b    '{'
dc.b    '}'
dc.b    -1      ;<<<<<<<<<<<<<<<<<<
dc.b    '1'     ;tastierino
dc.b    '2'     ;tastierino
dc.b    '3'     ;tastierino
dc.b    'A'
dc.b    'S'
dc.b    'D'
dc.b    'F'
dc.b    'G'
dc.b    'H'
dc.b    'J'
dc.b    'K'
dc.b    'L'
dc.b    ':'
dc.b    '"'
dc.b    -1      ;not used
dc.b    -1      ;<<<<<<<<<<<<<<<<<<<<<<
dc.b    '4'     ;tastierino
dc.b    '5'     ;tastierino
dc.b    '6'     ;tastierino
dc.b    '>'
dc.b    'Z'
dc.b    'X'
dc.b    'C'
dc.b    'V'
dc.b    'B'
dc.b    'N'
dc.b    'M'
dc.b    '<'
dc.b    '>'
dc.b    '?'
dc.b    -1      ;<<<<<<<<<<<<<<<<<<<
dc.b    '.'     ;tastierino
dc.b    '7'     ;tastierino
dc.b    '8'     ;tastierino
dc.b    '9'     ;tastierino
dc.b    ' '     ;space
dc.b    -1      ;back space
dc.b    -1      ;tab
dc.b    -1      ;return tastierino
dc.b    -1      ;return
dc.b    -1      ;esc
dc.b    -1      ;del
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;<<<<<<<<<
dc.b    '-'
```

```
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;up
dc.b    -1      ;down
dc.b    -1      ;dx
dc.b    -1      ;sx
dc.b    -1      ;f1
dc.b    -1      ;f2
dc.b    -1      ;f3
dc.b    -1      ;f4
dc.b    -1      ;f5
dc.b    -1      ;f6
dc.b    -1      ;f7
dc.b    -1      ;f8
dc.b    -1      ;f9
dc.b    -1      ;f10
dc.b    '('
dc.b    ')'
dc.b    '/'
dc.b    '*'
dc.b    '+'
dc.b    -1      ;help
dc.b    -1      ;lshift
dc.b    -1      ;rshift
dc.b    -1      ;caps lock
dc.b    -1      ;ctrl
dc.b    -1      ;lalt
dc.b    -1      ;ralt
dc.b    -1      ;lamiga
dc.b    -1      ;ramiga

dc.b    '`'     ;alt-tati
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    '.'
dc.b    '«'
dc.b    '»'
dc.b    '-'
dc.b    '='
dc.b    '\'
dc.b    -1      ;<<<<<<<<<<<<<
dc.b    '0'     ;tastierino numerico
dc.b    'å'
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    'ⓒ'
dc.b    'Ⓡ'
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
```

```
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    '¡'
dc.b    'ø'
dc.b    '¶'
dc.b    '['
dc.b    ']'
dc.b    -1      ;<<<<<<<<<<<<<<<<<
dc.b    '1'     ;tastierino
dc.b    '2'     ;tastierino
dc.b    '3'     ;tastierino
dc.b    'æ'
dc.b    'ß'
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ''
dc.b    ''
dc.b    ''
dc.b    ''
dc.b    ''
dc.b    '£'
dc.b    ';'
dc.b    39
dc.b    'ù'     ;not used
dc.b    -1      ;<<<<<<<<<<<<<<<<<<<<
dc.b    '4'     ;tastierino
dc.b    '5'     ;tastierino
dc.b    '6'     ;tastierino
dc.b    '<'
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    'ç'
dc.b    'ª'
dc.b    'º'
dc.b    ' '     ;questo non è visualizzabile qui,
                ;controlla i sorgenti testuali
dc.b    '¸'
dc.b    ','
dc.b    '.'
dc.b    '/'
dc.b    -1      ;<<<<<<<<<<<<<<<<<<
dc.b    '.'     ;tastierino
dc.b    '7'     ;tastierino
dc.b    '8'     ;tastierino
dc.b    '9'     ;tastierino
dc.b    ' '     ;space
dc.b    -1      ;back space
dc.b    -1      ;tab
dc.b    -1      ;return tastierino
dc.b    -1      ;return
dc.b    '>'     ;esc
dc.b    -1      ;del
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;<<<<<<<<<
dc.b    '-'
dc.b    -1      ;<<<<<<<<<
dc.b    -1      ;up
dc.b    -1      ;down
dc.b    -1      ;dx
```

```
        dc.b    -1      ;sx
        dc.b    -1      ;f1
        dc.b    -1      ;f2
        dc.b    -1      ;f3
        dc.b    -1      ;f4
        dc.b    -1      ;f5
        dc.b    -1      ;f6
        dc.b    -1      ;f7
        dc.b    -1      ;f8
        dc.b    -1      ;f9
        dc.b    -1      ;f10
        dc.b    '['
        dc.b    ']'
        dc.b    '/'
        dc.b    '*'
        dc.b    '+'
        dc.b    -1      ;help
        dc.b    -1      ;lshift
        dc.b    -1      ;rshift
        dc.b    -1      ;caps lock
        dc.b    -1      ;ctrl
        dc.b    -1      ;lalt
        dc.b    -1      ;ralt
        dc.b    -1      ;lamiga
        dc.b    -1      ;ramiga

        even

;       Il FONT caratteri 8x8.

FONT:
        incbin  "assembler2:sorgenti4/nice.fnt"

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8e,$2c81       ; DiwStrt       (registri con valori normali)
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$003c       ; DdfStart HIRES
        dc.w    $94,$00d4       ; DdfStop HIRES
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod \ INTERLACE: modulo = lungh. linea!
        dc.w    $10a,0          ; Bpl2Mod / per saltarle (le pari o le disp.)

                ; 5432109876543210
        dc.w    $100,%1001001000000000  ; 1 bitplane, HIRES 640x256

BPLPOINTERS:
        dc.w $e0,$0000,$e2,$0000        ;primo   bitplane

        dc.w    $180,$226       ; color0 - SFONDO
        dc.w    $182,$0c0       ; color1 - plane 1 posizione normale, e'
                                ; la parte che "sporge" in alto.

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
*****************************************************************************
;                                MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.fairlight"

;*****************************************************************
;       Il bitplane
;*****************************************************************
        section bitplane,bss_C

BITPLANE:
        ds.b    80*320

        end
```

Potreste farvi una utility, oppure un programma che richiede l'immissione del nome o di altri dati, o uno che vi risponde a tono, come fosse una persona... fate voi!

27.12 Lezione11n1

```
; Lezione11n1.s -Routine di temporizzazione che permette di attendere un
;                certo numero di microsecondi usando un timer A del CIAA/B

; Questa routine di test permette di verificare a quante linee video
; corrispondono un certo un certo numero di microsecondi.
; (la parte ROSSA dello schermo e' quella in cui viene eseguita la routine)


MICS:   equ     2000            ; ~2000 microsecondi = ~2 millisecondi
                                ; valore = mics/1,4096837
                                ; 1 microsecondo = 1 sec/1 milione
                                ; NOTA: per raffrontare questa routine con
                                ; quella che attende le linee raster, fate
                                ; conto che 200 millisecondi corrispondono
                                ; circa a 5 linee di raster, 400 millisecondi
                                ; a 9,5 linee, 600 millis. a 14 linee eccetera

Start:
        move.l  4.w,a6          ; Execbase in a6
        jsr     -$84(a6)        ; forbid
        jsr     -$78(a6)        ; disable
        LEA     $DFF000,A5

WBLANNY:
        MOVE.L  4(A5),D0        ; $dff004 - VPOSR/VHPOSR
        ANDI.L  #$1FF00,D0      ; con interessano solo i bit della linea vert.
        CMPI.L  #$08000,D0      ; aspetta la linea $080
        BNE.S   WBLANNY

        move.w  #$f00,$180(a5)  ; Colore zero ROSSO

        bsr.s   CIAMIC
```

```
        move.w  #$0f0,$180(a5)  ; Colore zero VERDE

        btst    #6,$bfe001
        bne.s   WBLANNY


        move.l  4.w,a6          ; Execbase in a6
        jsr     -$7e(a6)        ; enable
        jsr     -$8a(a6)        ; permit
        rts

;       Ecco la routine che aspetta un numero specifico di MICROSECONDI,
;       usando il timer A del CIAB. Per usare il timer A del CIAA basta
;       sostituire il "lea $bfd000,a4" con un "lea $bfe001,a4". Nel listato
;       e' gia' presente, basta togliere il punto e virgola, e metterlo
;       invece al CIAB base. Comunque e' meglio usare il CIAB perche' il
;       timer CIAA e' usato dal sistema operativo per vari compiti.

CIAMIC:
        movem.l d0/a4,-(sp)             ; salviamo i registri usati
        lea     $bfd000,a4              ; CIAB base
        lea     $bfe001,a4              ; CIAA base (se volete usare il B)
        move.b  $e00(a4),d0             ; $bfde00 - CRA, CIAB control reg. A
        andi.b   #%11000000,d0          ; azzera i bit 0-5
        ori.b    #%00001000,d0          ; One-Shot mode (runmode singolo)
        move.b  d0,$e00(a4)             ; CRA - Setta il registro di controllo
        move.b  #%01111110,$d00(a4)     ; ICR - cancella gli interrupts CIA
        move.b  #(MICS&$FF),$400(a4)    ; TALO - metti il byte basso del time
        move.b  #(MICS>>8),$500(a4)     ; TAHI - metti il byte alto del time
        bset.b  #0,$e00(a4)             ; CRA - Start timer!!
wait:
        btst.b  #0,$d00(a4)     ; ICR - Attendiamo che il tempo sia scaduto
        beq.s   wait
        movem.l (sp)+,d0/a4             ; ripristiniamo i registri
        rts

        end

; CIA:  ICR  (Interrupt Control Register)                               [d]
;
; 0     TA              underflow
; 1     TB              underflow
; 2     ALARM           TOD alarm
; 3     SP              serial port full/empty
; 4     FLAG            flag
; 5-6   unused
; 7  R  IR
; 7  W  set/clear
;
; CIA:  CRA, CRB  (Control Register)                                    [e-f]
;
; 0     START           0 = stop / 1 = start TA; {0}=0 when TA underflow
; 1     PBON            1 = TA output on PB / 0 = normal mode
; 2     OUTMODE         1 = toggle / 0 = pulse
; 3     RUNMODE         1 = one-shot / 0 = continous mode
; 4  S  LOAD            1 = force load (strobe, always 0)
; 5   A INMODE          1 = TA counts positive CNT transition
;                       0 = TA counts 02 pulses
; 6   A SPMODE          serial port....
; 7   A unused
; 6-5 B INMODE          00 = TB counts 02 pulses
;                       01 = TB counts positive CNT transition
```

```
;                       10 = TB counts TA underflow pulses
;                       11 = TB counts TA underflow pulses while CNT is high
; 7   B ALARM           1 = writing TOD sets alarm
;                       0 = writing TOD sets clock
;                       Reading TOD always reads TOD clock
```

Lezione11n1b

```
; Lezione11n1.s -Routine di temporizzazione che permette di attendere un
;                certo numero di microsecondi usando un timer B del CIAA/B

; Questa routine di test permette di verificare a quante linee video
; corrispondono un certo un certo numero di microsecondi.
; (la parte ROSSA dello schermo e' quella in cui viene eseguita la routine)


MICS:   equ     2000            ; ~2000 microsecondi = ~2 millisecondi
                                ; valore = mics/1,4096837
                                ; 1 microsecondo = 1 sec/1 milione
                                ; NOTA: per raffrontare questa routine con
                                ; quella che attende le linee raster, fate
                                ; conto che 200 millisecondi corrispondono
                                ; circa a 5 linee di raster, 400 millisecondi
                                ; a 9,5 linee, 600 millis. a 14 linee eccetera

Start:
        move.l  4.w,a6          ; Execbase in a6
        jsr     -$84(a6)        ; forbid
        jsr     -$78(a6)        ; disable
        LEA     $DFF000,A5

WBLANNY:
        MOVE.L  4(A5),D0        ; $dff004 - VPOSR/VHPOSR
        ANDI.L  #$1FF00,D0      ; con interessano solo i bit della linea vert.
        CMPI.L  #$08000,D0      ; aspetta la linea $080
        BNE.S   WBLANNY

        move.w  #$f00,$180(a5)  ; Colore zero ROSSO

        bsr.s   CIAMIC

        move.w  #$0f0,$180(a5)  ; Colore zero VERDE

        btst    #6,$bfe001
        bne.s   WBLANNY


        move.l  4.w,a6          ; Execbase in a6
        jsr     -$7e(a6)        ; enable
        jsr     -$8a(a6)        ; permit
        rts

;       Ecco la routine che aspetta un numero specifico di MICROSECONDI,
;       usando il timer B del CIAB. Per usare il timer B del CIAA basta
;       sostituire il "lea $bfd000,a4" con un "lea $bfe001,a4". Nel listato
;       e' gia' presente, basta togliere il punto e virgola, e metterlo
;       invece al CIAB base. Comunque e' meglio usare il CIAB perche' il
;       timer CIAA e' usato dal sistema operativo per vari compiti.
;       Infatti, se usate il timerB del ciaA, si blocca tutto!
```

```
CIAMIC:
        movem.l d0/a4,-(sp)             ; salviamo i registri usati
        lea     $bfd000,a4              ; CIAB base

;       lea     $bfe001,a4              ; CIAA base - PERO' MANDA IN BLOCCO
                                        ; TUTTO, DATO CHE LO USA IL SISTEMA
                                        ; OPERATIVO! NON USATE IL TIMER B
                                        ; DEL CIAA, PER FAVORE!

        move.b  $f00(a4),d0             ; $bfde00 - CRB, CIAB control reg. B
        andi.b   #%11000000,d0          ; azzera i bit 0-5
        ori.b    #%00001000,d0          ; One-Shot mode (runmode singolo)
        move.b  d0,$f00(a4)             ; CRB - Setta il registro di controllo
        move.b  #%01111101,$d00(a4)     ; ICR - cancella gli interrupts CIA
        move.b  #(MICS&$FF),$600(a4)    ; TBLO - metti il byte basso del time
        move.b  #(MICS>>8),$700(a4)     ; TBHI - metti il byte alto del time
        bset.b  #0,$f00(a4)             ; CRB - Start timer!!
wait:
        btst.b  #1,$d00(a4)     ; ICR - Attendiamo che il tempo sia scaduto.
                                ; da notare che si testa il bit 1, e non lo
                                ; zero, per attendere il timer B.
        beq.s   wait
        movem.l (sp)+,d0/a4             ; ripristiniamo i registri
        rts

        end

Solo un'ultima cosa. Se eveste il tempo da aspettare in una label, potreste
"spezzettarlo" in byte basso e byte alto con un lsr:

        lea     $bfd000,a4              ; cia_b base
        move.w  TimerValue(PC),d0       ; countdown
        move.b  d0,$600(a4)             ; timer B - set lo byte
        lsr.w   #8,d0
        move.b  d0,$700(a4)             ; timer B - set hi byte
                ; 76543210
        move.b  #%01111101,$d00(a4)     ; ICR - cancella gli interrupts CIA
        move.b  #%00011001,$f00(a4)     ; CRB - start
                                        ; 7 - Alarm -> 0
                                        ; 6,5 - Inmode bits -> 00
                                        ; 4 - Load bit -> 1 (fa caricare al
                                        ;     timer il valore, e comincia
                                        ;     il conto alla rovescia).
                                        ; 3 - RunMode -> 1 (One shot, 1 volta)
                                        ; 2 - OutMode -> 0 (per ricev. pulse)
                                        ; 1 - PBON -> 0
                                        ; 0 - Start -> comuncia il conto alla
                                        ;     rovescia; giuto a 0->interrupt
        MOVE.B  #%10000010,$d00(a4)     ; ICR - Abilita interr. timer B ciaB
loop:
        btst    #1,$d00(a4)             ; ICR - test tb-bit ->clear ICR
        beq.s   loop                    ; not set->wait
        rts


; CIA:  ICR  (Interrupt Control Register)                               [d]
;
; 0     TA              underflow
; 1     TB              underflow
; 2     ALARM           TOD alarm
; 3     SP              serial port full/empty
; 4     FLAG            flag
```

```
; 5-6   unused
; 7  R  IR
; 7  W  set/clear
;
; CIA:  CRA, CRB  (Control Register)                                    [e-f]
;
; 0     START           0 = stop / 1 = start TA; {0}=0 when TA underflow
; 1     PBON            1 = TA output on PB / 0 = normal mode
; 2     OUTMODE         1 = toggle / 0 = pulse
; 3     RUNMODE         1 = one-shot / 0 = continous mode
; 4  S  LOAD            1 = force load (strobe, always 0)
; 5   A INMODE          1 = TA counts positive CNT transition
;                       0 = TA counts 02 pulses
; 6   A SPMODE          serial port....
; 7   A unused
; 6-5 B INMODE          00 = TB counts 02 pulses
;                       01 = TB counts positive CNT transition
;                       10 = TB counts TA underflow pulses
;                       11 = TB counts TA underflow pulses while CNT is high
; 7   B ALARM           1 = writing TOD sets alarm
;                       0 = writing TOD sets clock
;                       Reading TOD always reads TOD clock
```

Lezione11n2

```
; Lezione11n2.s -Routine di temporizzazione che permette di attendere un
;                certo numero di Hertz

Start:
        move.l  4.w,a6          ; Execbase in a6
        jsr     -$84(a6)        ; forbid
        jsr     -$78(a6)        ; disable
        LEA     $DFF000,A5

        bsr.s   CIAHZ   ; Attendi un paio di secondi

        move.l  4.w,a6          ; Execbase in a6
        jsr     -$7e(a6)        ; enable
        jsr     -$8a(a6)        ; permit
        rts


; bfe801 todlo  -       1=~0,02 secs o 1/50 sec (PAL) o 1/60 sec (NTSC)
; bfe901 todmid -       1=~3 secs
; bfea01 todhi  -       1=~21 mins
;
; In pratica e' un timer che puo' contenere un numero a 23 bit, e tale num.
; e' diviso: bits 0-7 in TODLO, bits 8-15 in TODMID e bits 16-23 in TODHI.


CIAHZ:
        MOVE.L  A2,-(SP)
        LEA     $BFE001,A2      ; CIAA base -> USATO
;       LEA     $BFD000,A2      ; CIAB base

        MOVE.B  #0,$800(A2)     ; TODLO - bit 7-0 del timer a 50-60hz
                                ; reset timer!
WCIA:
        CMPI.B  #50*2,$800(A2)  ; TODLO - Wait time = 2 secondi...
        BGE.S   DONE
```

```
        BRA.S   WCIA
DONE:
        MOVE.L  (SP)+,A2
        RTS

        end
```

Da notare che se si vuole usare il CIAB, si passa ad un timer di sync orizzontale e non verticale, per cui e' mooolto piu' veloce. Per attendere 2 secondi circa occorre scomodare il TODMID:

```
CIAHZ:
        MOVE.L  A2,-(SP)
;       LEA     $BFE001,A2      ; CIAA base
        LEA     $BFD000,A2      ; CIAB base -> USATO

        MOVE.B  #0,$800(A2)     ; TODLO - bit 7-0 del timer a 50-60hz
                                ; reset timer!
WCIA:
        CMPI.B  #120,$900(A2)   ; TODMID - Wait time = 2 secondi...
        BGE.S   DONE
        BRA.S   WCIA
DONE:
        MOVE.L  (SP)+,A2
        RTS
```

Attenzione al fatto che il TOD del CIAA e' usato dal timer.device, mentre il TOD del CIAB e' usato dalla graphics.library!

Se potete, aspettate tempi brevi con la classica routine:

```
        lea     $dff006,a0      ; VHPOSR
        moveq   #XXX-1,d0       ; Numero di linee da aspettare
waitlines:
        move.b  (a0),d1         ; $dff006 - linea verticale attuale in d1
stepline:
        cmp.b   (a0),d1         ; siamo sempre alla stessa linea?
        beq.s   stepline        ; se si aspetta
        dbra    d0,waitlines    ; linea "aspettata", aspetta d0-1 linee
```

27.13 Lezione11o1

```
; Lezione11o1.s   Caricamento di un file dati usando la dos.library

        Section DosLoad,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

Maincode:
        movem.l d0-d7/a0-a6,-(SP)       ; Salva i registri nello stack
        move.l  4.w,a6                  ; ExecBase in a6
        LEA     DosName(PC),A1          ; Dos.library
        JSR     -$198(A6)               ; OldOpenlib
        MOVE.L  D0,DosBase
        BEQ.s   EXIT                    ; Se zero, esci! Errore!

Mouse:
        btst.b  #6,$bfe001      ; ciaapra - tasto sin. del mouse
        bne.s   Mouse
```

```
        bsr.s   CaricaFile      ; Carica un file con la dos.library

        MOVE.L  DosBase(PC),A1  ; DosBase in A1 per chiudere la libreria
        move.l  4.w,a6          ; ExecBase in A6
        jsr     -$19E(a6)       ; CloseLibrary - dos.library CHIUSA
EXIT:
        movem.l (SP)+,d0-d7/a0-a6 ; Riprendi i vecchi valori dei registri
        RTS                       ; Torna all'ASMONE o al Dos/WorkBench


DosName:
        dc.b    "dos.library",0
        even

DosBase:                ; Puntatore alla Base della Dos Library
        dc.l    0

*****************************************************************************
; Routine che carica un file di una lunghezza specificata e con un nome
; specificato. Occorre mettere l'intero path, se questo esiste!
*****************************************************************************

CaricaFile:
        move.l  #filename,d1    ; indirizzo con stringa "file name + path"
        MOVE.L  #$3ED,D2        ; AccessMode: MODE_OLDFILE - File che esiste
                                ; gia', e che quindi potremo leggere.
        MOVE.L  DosBase(PC),A6
        JSR     -$1E(A6)        ; LVOOpen - "Apri" il file
        MOVE.L  D0,FileHandle   ; Salva il suo handle
        BEQ.S   ErrorOpen       ; Se d0 = 0 allora c'e' un errore!

        MOVE.L  D0,D1           ; FileHandle in d1 per il Read
        MOVE.L  #buffer,D2      ; Indirizzo Destinazione in d2
        MOVE.L  #42240,D3       ; Lunghezza del file (ESATTA!)
        MOVE.L  DosBase(PC),A6
        JSR     -$2A(A6)        ; LVORead - leggi il file e copialo nel buffer

        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
        JSR     -$24(A6)        ; LVOClose - chiudi il file.
ErrorOpen:
        rts


FileHandle:
        dc.l    0

; Stringa di testo, da terminare con uno 0, a cui dovra' puntare d1 prima di
; fare l'OPEN della dos.lib. Conviene mettere l'intero path.

Filename:
        dc.b    "assembler2:sorgenti7/amiet.raw",0      ; path+nomefile
        even

*****************************************************************************
; Buffer dove viene caricata l'immagine da disco (o hard disk) tramite doslib
*****************************************************************************

        section mioplanaccio,bss

buffer:
LOGO:
```

```
        ds.b    6*40*176        ; 6 bitplanes * 176 lines * 40 bytes (HAM)

        end
```

Lezione11o2

```
; Lezione11o2.s   Caricamento di un file dati usando la dos.library
;                 Premere il tasto sinistro per caricare, destro per uscire

        Section DosLoad,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper,bitplane DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

; Puntiamo la PIC

        MOVE.L  #PICTURE2,d0
        LEA     BPLPOINTERS2,A1
        MOVEQ   #5-1,D1                 ; num di bitplanes
POINTBT2:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #34*40,d0               ; lunghezza del bitplane
        addq.w  #8,a1
        dbra    d1,POINTBT2     ; Rifai D1 volte (D1=num do bitplanes)

; Puntiamo la PIC che sara' caricata (ora e' solo un buffer vuoto)

        LEA     bplpointers,A0
        MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
        MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
        ADDQ.w  #8,A0
        ADD.L   #176*40,D0      ; lunghezza plane
        DBRA    D7,pointloop


; Puntiamo il nostro int di livello 3

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  oldint6c(PC),crappyint  ; Per DOS LOAD - salteremo all'oldint
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
```

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$c020,$9a(a5)  ; INTENA - abilito interrupt "VERTB"
                                ; del livello 3 ($6c)

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   DosLoad         ; Carica un file legalmente con la dos.lib
                                ; mentre stiamo visualizzando una nostra
                                ; copperlist e eseguendo un nostro interrupt
        TST.L   ErrorFlag
        bne.s   ErroreLoad      ; File non caricato?? Non usiamolo allora!

mouse2:
        btst    #2,$dff016      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse2          ; loop in modo utente, e ogni vertical blank

ErroreLoad:
        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        bsr.w   ColorCicla              ; Cicla i colori della pic
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
NointVERTB:
        move.w  #$70,$dff09c    ; INTREQ - int eseguito, cancello la richiesta
                                ; dato che il 680x0 non la cancella da solo!!!
        rte                     ; Uscita dall'int VERTB

*****************************************************************************
*       Routine che "cicla" i colori di tutta la palette.                   *
*       Questa routine cicla i primi 15 colori separatamente dal secondo    *
*       secondo blocco di colori. Funziona come i "RANGE" del Dpaint.       *
*****************************************************************************

;       Il contatore "cont" serve a far aspettare 3 fotogrammi prima di
;       eseguire la routine cont. In pratica a "rallentare" l'esecuzione
```

```
cont:
        dc.w    0

ColorCicla:
        addq.b  #1,cont
        cmp.b   #3,cont         ; Agisci 1 volta ogni 3 fotogrammi solamente
        bne.s   NonAncora       ; Non siamo ancora al terzo? Esci!
        clr.b   cont            ; Siamo al terzo, azzera il contatore

; Roteazione all'indietro dei primi 15 colori

        lea     cols+2,a0       ; Indirizzo primo colore del primo gruppo
        move.w  (a0),d0         ; Salva il primo colore in d0
        moveq   #15-1,d7        ; 15 colori da "roteare" nel primo gruppo
cloop1:
        move.w  4(a0),(a0)      ; Copia il colore avanti in quello prima
        addq.w  #4,a0           ; salta alla prossimo col. da "indietreggiare"
        dbra    d7,cloop1       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema il primo colore salvato come ultimo.

; Roteazione in avanti dei secondi 15 colori

        lea     cole-2,a0       ; Indirizzo ultimo colore del secondo gruppo
        move.w  (a0),d0         ; Salva l'ultimo colore in d0
        moveq   #15-1,d7        ; Altri 15 colori da "roteare" separatamente
cloop2:
        move.w  -4(a0),(a0)     ; Copia il colore indietro in quello dopo
        subq.w  #4,a0           ; salta al precedente col. da "avanzare"
        dbra    d7,cloop2       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema l'ultimo colore salvato come primo
NonAncora:
        rts


*****************************************************************************
; Routine che carica un file mentre stiamo battendo nel metallo.
*****************************************************************************

DosLoad:
        bsr.w   PreparaLoad     ; Rispristina multitask e setta interrupt load

        moveq   #5,d1           ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 5 frames

        bsr.s   CaricaFile      ; Carica il file con la dos.library
        move.l  d0,ErrorFlag    ; Salva lo stato di successo o di errore

; nota: ora dobbiamo attendere che il motore del disk drive, o la spia del
; povero Hard Disk o CD-ROM si spenga, prima di bloccare tutto, o causiamo
; un crash del sistema spettacolare.

        move.w  #150,d1         ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 150 frames

        bsr.w   DopoLoad        ; Disabilita multitask e rimetti interrupt
        rts

ErrorFlag:
        dc.l    0

*****************************************************************************
; Routine che carica un file di una lunghezza specificata e con un nome
```

```
; specificato. Occorre mettere l'intero path!
*****************************************************************************

CaricaFile:
        move.l  #filename,d1    ; indirizzo con stringa "file name + path"
        MOVE.L  #$3ED,D2        ; AccessMode: MODE_OLDFILE - File che esiste
                                ; gia', e che quindi potremo leggere.
        MOVE.L  DosBase(PC),A6
        JSR     -$1E(A6)        ; LVOOpen - "Apri" il file
        MOVE.L  D0,FileHandle   ; Salva il suo handle
        BEQ.S   ErrorOpen       ; Se d0 = 0 allora c'e' un errore!

; Carichiamo il file

        MOVE.L  D0,D1           ; FileHandle in d1 per il Read
        MOVE.L  #buffer,D2      ; Indirizzo Destinazione in d2
        MOVE.L  #42240,D3       ; Lunghezza del file (ESATTA!)
        MOVE.L  DosBase(PC),A6
        JSR     -$2A(A6)        ; LVORead - leggi il file e copialo nel buffer
        cmp.l   #-1,d0          ; Errore? (qua e' indicato con -1)
        beq.s   ErroreRead

; Chiudiamolo

        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
        JSR     -$24(A6)        ; LVOClose - chiudi il file.

; Attenzione al fatto che se non CHIUDETE il file, gli altri programmi non
; potranno accedere a tale file (non potrete ne' cancellarlo, ne' spostarlo).

        moveq   #0,d0   ; Segnaliamo il successo
        rts

; Qua ci son le dolenti note, in caso di errore:

ErroreRead:
        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
        JSR     -$24(A6)        ; LVOClose - chiudi il file.
ErrorOpen:
        moveq   #-1,d0  ; segnaliamo l'insuccesso
        rts


FileHandle:
        dc.l    0

; Stringa di testo, da terminare con uno 0, a cui dovra' puntare d1 prima di
; fare l'OPEN della dos.lib. Conviene mettere l'intero path.

Filename:
        dc.b    "assembler2:sorgenti7/amiet.raw",0      ; path+nomefile
        even

*****************************************************************************
; Routine di interrupt da mettere durante il caricamento. Le routines che
; saranno messe in questo interrupt saranno eseguite anche durante il
; caricamento, sia che avvenga da floppy disk, da Hard Disk, o CD ROM.
; DA NOTARE CHE STIAMO USANDO L'INTERRUPT COPER, E NON QUELLO VBLANK,
; QUESTO PERCHE' DURANTE IL CARICAMENTO DA DISCO, SPECIALMENTE SOTTO KICK 1.3,
; L'INTERRUPT VERTB NON E' STABILE, tanto che la musica avrebbe dei sobbalzi.
```

```
; Invece, se mettiamo un "$9c,$8010" nella nostra copperlist, siamo sicuri
; che questa routine sara' eseguita una volta sola per fotogramma.
*****************************************************************************

myint6cLoad:
        btst.b  #4,$dff01f      ; INTREQR - il bit 4, COPER, e' azzerato?
        beq.s   nointL          ; Se si, non e' un "vero" int COPER!
        move.w  #%10000,$dff09c ; Se no, e' la volta buona, togliamo il req!
        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_music        ; Suona la musica
        movem.l (SP)+,d0-d7/a0-a6
nointL:
        dc.w    $4ef9           ; val esadecimale di JMP
Crappyint:
        dc.l    0               ; Indirizzo dove Jumpare, da AUTOMODIFICARE...
                                ; ATTENZIONE: il codice automodificante non
                                ; andrebbe usato. Comunque se si chiama un
                                ; ClearMyCache prima e dopo, funziona!

*****************************************************************************
; Routine che ripristina il sistema operativo, tranne la copperlist, e in
; piu' setta un interrupt $6c nostro, che poi salta a quello di sistema.
; Da notare che durante il load l'interrupt e' gestito dall'int "COPER"
*****************************************************************************

PreparaLoad:
        LEA     $DFF000,A5              ; Base dei registri CUSTOM per Offsets
        MOVE.W  $2(A5),OLDDMAL          ; Salva il vecchio status di DMACON
        MOVE.W  $1C(A5),OLDINTENAL      ; Salva il vecchio status di INTENA
        MOVE.W  $1E(A5),OLDINTREQL      ; Salva il vecchio status di INTREQ
        MOVE.L  #$80008000,d0           ; Prepara la maschera dei bit alti
        OR.L    d0,OLDDMAL              ; Setta il bit 15 dei valori salvati
        OR.W    d0,OLDINTREQL           ; dei registri, per poterli rimettere.

        bsr.w   ClearMyCache

        MOVE.L  #$7FFF7FFF,$9A(a5)      ; DISABILITA GLI INTERRUPTS & INTREQS

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  OldInt64(PC),$64(a0) ; Sys int liv1 salvato (softint,dskblk)
        move.l  OldInt68(PC),$68(a0) ; Sys int liv2 salvato (I/O,ciaa,int2)
        move.l  #myint6cLoad,$6c(a0) ; Int che poi salta a quello di sys.
        move.l  OldInt70(PC),$70(a0) ; Sys int liv4 salvato (audio)
        move.l  OldInt74(PC),$74(a0) ; Sys int liv5 salvato (rbf,dsksync)
        move.l  OldInt78(PC),$78(a0) ; Sys int liv6 salvato (exter,ciab,inten)

        MOVE.W  #%1000001001010000,$96(A5) ; Abilita blit e disk per sicurezza
        MOVE.W  OLDINTENA(PC),$9A(A5)   ; INTENA STATUS
        MOVE.W  OLDINTREQ(PC),$9C(A5)   ; INTREQ
        move.w  #$c010,$9a(a5)          ; dobbiamo essere sicuri che COPER
                                        ; (interrupt via copperlist) sia ON!

        move.l  4.w,a6
        JSR     -$7e(A6)        ; Enable
        JSR     -$8a(a6)        ; Permit

        MOVE.L  GfxBase(PC),A6
        jsr     -$E4(A6)        ; Aspetta la fine di eventuali blittate
        JSR     -$E4(A6)        ; WaitBlit
        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
                                ; (nel kick 1.3 serve per caricare da disk)
```

```
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #0,D0           ; Priorita' in d0 (-128, +127) - NORMALE
                                ; (Per permettere ai drives di respirare)
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)
        rts

OLDDMAL:
        dc.w    0
OLDINTENAL:             ; Vecchio status INTENA
        dc.w    0
OLDINTREQL:             ; Vecchio status INTREQ
        DC.W    0

*****************************************************************************
; Routine che richiude il sistema operativo e rimette il nostro interrupt
*****************************************************************************

DopoLoad:
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #127,D0         ; Priorita' in d0 (-128, +127) - MASSIMA
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)

        JSR     -$84(a6)        ; Forbid
        JSR     -$78(A6)        ; Disable

        MOVE.L  GfxBase(PC),A6
        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.
        jsr     -$E4(A6)        ; WaitBlit - Attende la fine di ogni blittata
        JSR     -$E4(A6)        ; WaitBlit

        bsr.w   ClearMyCache

        LEA     $dff000,a5              ; Custom base per offsets
AspettaF:
        MOVE.L  4(a5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        AND.L   #$1ff00,D0      ; Seleziona solo i bit della pos. verticale
        CMP.L   #$12d00,D0      ; aspetta la linea $12d per evitare che
        BEQ.S   AspettaF        ; spegnendo i DMA si abbiano sfarfallamenti

        MOVE.L  #$7FFF7FFF,$9A(A5)      ; DISABILITA GLI INTERRUPTS & INTREQS

                ; 5432109876543210
        MOVE.W  #%0000010101110000,d0   ; DISABILITA DMA

        btst    #8-8,olddmal    ; test bitplane
        beq.s   NoPlanesA
        bclr.l  #8,d0           ; non spengere planes
NoPlanesA:
        btst    #5,olddmal+2    ; test sprite
        beq.s   NoSpritez
        bclr.l  #5,d0           ; non spengere sprite
NoSpritez:
        MOVE.W  d0,$96(A5) ; DISABILITA DMA

        move.l  BaseVBR(PC),a0          ; In a0 il valore del VBR
```

```
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        MOVE.W  OLDDMAL(PC),$96(A5)     ; Rimetti il vecchio status DMA
        MOVE.W  OLDINTENAL(PC),$9A(A5)  ; INTENA STATUS
        MOVE.W  OLDINTREQL(PC),$9C(A5)  ; INTREQ
        rts

*****************************************************************************
; Questa routine aspetta D1 fotogrammi. Ogni 50 fotogrammi passa 1 secondo.
;
; d1 = numero di fotogrammi da attendere
;
*****************************************************************************

AspettaBlanks:
        LEA     $DFF000,A5      ; CUSTOM REG per OFFSETS
WBLAN1xb:
        MOVE.w  #$80,D0
WBLAN1bxb:
        CMP.B   6(A5),D0        ; vhposr
        BNE.S   WBLAN1bxb
WBLAN2xb:
        CMP.B   6(A5),D0        ; vhposr
        Beq.S   WBLAN2xb
        DBRA    D1,WBLAN1xb
        rts

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero


                ;5432109876543210
        dc.w    $100,%0110101000000000  ; bplcon0 - 320*256 HAM!

        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
```

```
        dc.w $1a0,$0666

        dc.w    $9707,$FFFE     ; wait linea $97

        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $b907,$fffe     ; WAIT - attendi linea $b9
BPLPOINTERS2:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "

        dc.w    $100,%0101001000000000  ; BPLCON0 - 5 bitplanes LOWRES

; La palette, che sara' "ruotata" in 2 gruppi di 16 colori.

cols:
        dc.w $180,$040,$182,$050,$184,$060,$186,$080    ; tono verde
        dc.w $188,$090,$18a,$0b0,$18c,$0c0,$18e,$0e0
        dc.w $190,$0f0,$192,$0d0,$194,$0c0,$196,$0a0
        dc.w $198,$090,$19a,$070,$19c,$060,$19e,$040

        dc.w $1a0,$029,$1a2,$02a,$1a4,$13b,$1a6,$24b    ; tono blu
        dc.w $1a8,$35c,$1aa,$36d,$1ac,$57e,$1ae,$68f
        dc.w $1b0,$79f,$1b2,$68f,$1b4,$58e,$1b6,$37e
        dc.w $1b8,$26d,$1ba,$15d,$1bc,$04c,$1be,$04c
cole:

        dc.w    $da07,$fffe     ; WAIT - attendi la linea $da
        dc.w    $100,$200       ; BPLCON0 - disabilita i bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $ff07,$fffe     ; WAIT - attendi la linea $ff
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, per
                                ; suonare la musica (anche mentre stiamo
                                ; caricando con la dos.library).

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


*****************************************************************************
;               DISEGNO 320*34 a 5 bitplanes (32 colori)
*****************************************************************************

PICTURE2:
        INCBIN  "pic320*34*5.raw"

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.fairlight"

*****************************************************************************
; Buffer dove viene caricata l'immagine da disco (o hard disk) tramite doslib
```

```
*****************************************************************************

        section mioplanaccio,bss_C

buffer:
LOGO:
        ds.b    6*40*176        ; 6 bitplanes * 176 lines * 40 bytes (HAM)


        end
```

In questo esempio carichiamo il logo, che appare subito sopra. Se lo caricate da dischetto noterete che appare plane per plane, a pezzi, infatti carica un po' alla volta! Sarebbe meglio caricarlo in un buffer a parte, poi farlo visualizzare tutto insieme a caricamento avvenuto.
La cosa fondamentale del caricamento e' che li tempo atteso dopo il load, prima di richiudere tutto, sia sufficiente. Altrimenti e' la fine!
In Interrupt non viene eseguita la routine dei colori, ma solo quella della musica, almeno si nota il tempo che si attende "per sicurezza".
Dato che questo tempo va atteso comunque, sarebbe intelligente fare un fade o qualche routine che perde tempo facendo qualcosa di carino, prima di richiudere tutto, almeno si e' atteso il tempo, ma non senza usarlo!

Lezione11o2b

```
; Lezione11o2.s   Caricamento di un file dati usando la dos.library
;                 Premere il tasto sinistro per caricare, destro per uscire

        Section DosLoad,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper,bitplane DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

; Puntiamo la PIC

        MOVE.L  #PICTURE2,d0
        LEA     BPLPOINTERS2,A1
        MOVEQ   #5-1,D1                 ; num di bitplanes
POINTBT2:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #34*40,d0               ; lunghezza del bitplane
        addq.w  #8,a1
        dbra    d1,POINTBT2     ; Rifai D1 volte (D1=num do bitplanes)

; Puntiamo la PIC che sara' caricata (ora e' solo un buffer vuoto)
```

```
;       LEA     bplpointers,A0
;       MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
;       MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
;pointloop:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
        ADDQ.w  #8,A0
        ADD.L   #176*40,D0      ; lunghezza plane
;       DBRA    D7,pointloop


; Puntiamo il nostro int di livello 3

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  oldint6c(PC),crappyint  ; Per DOS LOAD - salteremo all'oldint
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        ;bsr.w  mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$c020,$9a(a5)  ; INTENA - abilito interrupt "VERTB"
                                ; del livello 3 ($6c)

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   DosLoad         ; Carica un file legalmente con la dos.lib
                                ; mentre stiamo visualizzando una nostra
                                ; copperlist e eseguendo un nostro interrupt
        TST.L   ErrorFlag
        bne.s   ErroreLoad      ; File non caricato?? Non usiamolo allora!

mouse2:
        btst    #2,$dff016      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse2          ; loop in modo utente, e ogni vertical blank

ErroreLoad:
        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
```

```
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        bsr.w   ColorCicla              ; Cicla i colori della pic
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
NointVERTB:
        move.w  #$70,$dff09c    ; INTREQ - int eseguito, cancello la richiesta
                                ; dato che il 680x0 non la cancella da solo!!!
        rte                     ; Uscita dall'int VERTB

*****************************************************************************
*       Routine che "cicla" i colori di tutta la palette.                   *
*       Questa routine cicla i primi 15 colori separatamente dal secondo    *
*       secondo blocco di colori. Funziona come i "RANGE" del Dpaint.       *
*****************************************************************************

;       Il contatore "cont" serve a far aspettare 3 fotogrammi prima di
;       eseguire la routine cont. In pratica a "rallentare" l'esecuzione

cont:
        dc.w    0

ColorCicla:
        addq.b  #1,cont
        cmp.b   #3,cont         ; Agisci 1 volta ogni 3 fotogrammi solamente
        bne.s   NonAncora       ; Non siamo ancora al terzo? Esci!
        clr.b   cont            ; Siamo al terzo, azzera il contatore

; Roteazione all'indietro dei primi 15 colori

        lea     cols+2,a0       ; Indirizzo primo colore del primo gruppo
        move.w  (a0),d0         ; Salva il primo colore in d0
        moveq   #15-1,d7        ; 15 colori da "roteare" nel primo gruppo
cloop1:
        move.w  4(a0),(a0)      ; Copia il colore avanti in quello prima
        addq.w  #4,a0           ; salta alla prossimo col. da "indietreggiare"
        dbra    d7,cloop1       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema il primo colore salvato come ultimo.

; Roteazione in avanti dei secondi 15 colori

        lea     cole-2,a0       ; Indirizzo ultimo colore del secondo gruppo
        move.w  (a0),d0         ; Salva l'ultimo colore in d0
        moveq   #15-1,d7        ; Altri 15 colori da "roteare" separatamente
cloop2:
        move.w  -4(a0),(a0)     ; Copia il colore indietro in quello dopo
        subq.w  #4,a0           ; salta al precedente col. da "avanzare"
        dbra    d7,cloop2       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema l'ultimo colore salvato come primo
NonAncora:
        rts


*****************************************************************************
; Routine che carica un file mentre stiamo battendo nel metallo.
*****************************************************************************

DosLoad:
        bsr.w   PreparaLoad     ; Rispristina multitask e setta interrupt load

        moveq   #5,d1           ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 5 frames
```

```
        bsr.s   CaricaFile      ; Carica il file con la dos.library
        move.l  d0,ErrorFlag    ; Salva lo stato di successo o di errore

        bsr.w   mt_init         ; Suona la musica caricata

; nota: ora dobbiamo attendere che il motore del disk drive, o la spia del
; povero Hard Disk o CD-ROM si spenga, prima di bloccare tutto, o causiamo
; un crash del sistema spettacolare.

        move.w  #150,d1         ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 150 frames

        bsr.w   DopoLoad        ; Disabilita multitask e rimetti interrupt
        rts

ErrorFlag:
        dc.l    0

*****************************************************************************
; Routine che carica un file di una lunghezza specificata e con un nome
; specificato. Occorre mettere l'intero path!
*****************************************************************************

CaricaFile:
        move.l  #filename,d1    ; indirizzo con stringa "file name + path"
        MOVE.L  #$3ED,D2        ; AccessMode: MODE_OLDFILE - File che esiste
                                ; gia', e che quindi potremo leggere.
        MOVE.L  DosBase(PC),A6
        JSR     -$1E(A6)        ; LVOOpen - "Apri" il file
        MOVE.L  D0,FileHandle   ; Salva il suo handle
        BEQ.S   ErrorOpen       ; Se d0 = 0 allora c'e' un errore!

; Carichiamo il file

        MOVE.L  D0,D1           ; FileHandle in d1 per il Read
        MOVE.L  #mt_data1,D2    ; Indirizzo Destinazione in d2
;       MOVE.L  D2,#buffer

        MOVE.L  #42240,D3       ; Lunghezza del file (ESATTA!)
        MOVE.L  DosBase(PC),A6
        JSR     -$2A(A6)        ; LVORead - leggi il file e copialo nel buffer
        cmp.l   #-1,d0          ; Errore? (qua e' indicato con -1)
        beq.s   ErroreRead

; Chiudiamolo

        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
        JSR     -$24(A6)        ; LVOClose - chiudi il file.

; Attenzione al fatto che se non CHIUDETE il file, gli altri programmi non
; potranno accedere a tale file (non potrete ne' cancellarlo, ne' spostarlo).

        moveq   #0,d0   ; Segnaliamo il successo
        rts

; Qua ci son le dolenti note, in caso di errore:

ErroreRead:
        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
```

```
        JSR     -$24(A6)        ; LVOClose - chiudi il file.
ErrorOpen:
        moveq   #-1,d0  ; segnaliamo l'insuccesso
        rts


FileHandle:
        dc.l    0

; Stringa di testo, da terminare con uno 0, a cui dovra' puntare d1 prima di
; fare l'OPEN della dos.lib. Conviene mettere l'intero path.

Filename:
        dc.b    "assembler2:sorgenti4/mod.fairlight",0  ; path+nomefile
        even

*****************************************************************************
; Routine di interrupt da mettere durante il caricamento. Le routines che
; saranno messe in questo interrupt saranno eseguite anche durante il
; caricamento, sia che avvenga da floppy disk, da Hard Disk, o CD ROM.
; DA NOTARE CHE STIAMO USANDO L'INTERRUPT COPER, E NON QUELLO VBLANK,
; QUESTO PERCHE' DURANTE IL CARICAMENTO DA DISCO, SPECIALMENTE SOTTO KICK 1.3,
; L'INTERRUPT VERTB NON E' STABILE, tanto che la musica avrebbe dei sobbalzi.
; Invece, se mettiamo un "$9c,$8010" nella nostra copperlist, siamo sicuri
; che questa routine sara' eseguita una volta sola per fotogramma.
*****************************************************************************

myint6cLoad:
        btst.b  #4,$dff01f      ; INTREQR - il bit 4, COPER, e' azzerato?
        beq.s   nointL          ; Se si, non e' un "vero" int COPER!
        move.w  #%10000,$dff09c ; Se no, e' la volta buona, togliamo il req!
        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_music        ; Suona la musica
        movem.l (SP)+,d0-d7/a0-a6
nointL:
        dc.w    $4ef9           ; val esadecimale di JMP
Crappyint:
        dc.l    0               ; Indirizzo dove Jumpare, da AUTOMODIFICARE...
                                ; ATTENZIONE: il codice automodificante non
                                ; andrebbe usato. Comunque se si chiama un
                                ; ClearMyCache prima e dopo, funziona!

*****************************************************************************
; Routine che ripristina il sistema operativo, tranne la copperlist, e in
; piu' setta un interrupt $6c nostro, che poi salta a quello di sistema.
; Da notare che durante il load l'interrupt e' gestito dall'int "COPER"
*****************************************************************************

PreparaLoad:
        LEA     $DFF000,A5              ; Base dei registri CUSTOM per Offsets
        MOVE.W  $2(A5),OLDDMAL          ; Salva il vecchio status di DMACON
        MOVE.W  $1C(A5),OLDINTENAL      ; Salva il vecchio status di INTENA
        MOVE.W  $1E(A5),OLDINTREQL      ; Salva il vecchio status di INTREQ
        MOVE.L  #$80008000,d0           ; Prepara la maschera dei bit alti
        OR.L    d0,OLDDMAL              ; Setta il bit 15 dei valori salvati
        OR.W    d0,OLDINTREQL           ; dei registri, per poterli rimettere.

        bsr.w   ClearMyCache

        MOVE.L  #$7FFF7FFF,$9A(a5)      ; DISABILITA GLI INTERRUPTS & INTREQS

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
```

```
        move.l  OldInt64(PC),$64(a0) ; Sys int liv1 salvato (softint,dskblk)
        move.l  OldInt68(PC),$68(a0) ; Sys int liv2 salvato (I/O,ciaa,int2)
        move.l  #myint6cLoad,$6c(a0) ; Int che poi salta a quello di sys.
        move.l  OldInt70(PC),$70(a0) ; Sys int liv4 salvato (audio)
        move.l  OldInt74(PC),$74(a0) ; Sys int liv5 salvato (rbf,dsksync)
        move.l  OldInt78(PC),$78(a0) ; Sys int liv6 salvato (exter,ciab,inten)

        MOVE.W  #%1000001001010000,$96(A5) ; Abilita blit e disk per sicurezza
        MOVE.W  OLDINTENA(PC),$9A(A5)   ; INTENA STATUS
        MOVE.W  OLDINTREQ(PC),$9C(A5)   ; INTREQ
        move.w  #$c010,$9a(a5)          ; dobbiamo essere sicuri che COPER
                                        ; (interrupt via copperlist) sia ON!

        move.l  4.w,a6
        JSR     -$7e(A6)        ; Enable
        JSR     -$8a(a6)        ; Permit

        MOVE.L  GfxBase(PC),A6
        jsr     -$E4(A6)        ; Aspetta la fine di eventuali blittate
        JSR     -$E4(A6)        ; WaitBlit
        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
                                ; (nel kick 1.3 serve per caricare da disk)
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #0,D0           ; Priorita' in d0 (-128, +127) - NORMALE
                                ; (Per permettere ai drives di respirare)
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)
        rts

OLDDMAL:
        dc.w    0
OLDINTENAL:             ; Vecchio status INTENA
        dc.w    0
OLDINTREQL:             ; Vecchio status INTREQ
        DC.W    0

*****************************************************************************
; Routine che richiude il sistema operativo e rimette il nostro interrupt
*****************************************************************************

DopoLoad:
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #127,D0         ; Priorita' in d0 (-128, +127) - MASSIMA
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)

        JSR     -$84(a6)        ; Forbid
        JSR     -$78(A6)        ; Disable

        MOVE.L  GfxBase(PC),A6
        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.
        jsr     -$E4(A6)        ; WaitBlit - Attende la fine di ogni blittata
        JSR     -$E4(A6)        ; WaitBlit

        bsr.w   ClearMyCache
```

```
        LEA     $dff000,a5              ; Custom base per offsets
AspettaF:
        MOVE.L  4(a5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        AND.L   #$1ff00,D0      ; Seleziona solo i bit della pos. verticale
        CMP.L   #$12d00,D0      ; aspetta la linea $12d per evitare che
        BEQ.S   AspettaF        ; spegnendo i DMA si abbiano sfarfallamenti

        MOVE.L  #$7FFF7FFF,$9A(A5)      ; DISABILITA GLI INTERRUPTS & INTREQS

                ; 5432109876543210
        MOVE.W  #%0000010101110000,d0   ; DISABILITA DMA

        btst    #8-8,olddmal    ; test bitplane
        beq.s   NoPlanesA
        bclr.l  #8,d0           ; non spengere planes
NoPlanesA:
        btst    #5,olddmal+2    ; test sprite
        beq.s   NoSpritez
        bclr.l  #5,d0           ; non spengere sprite
NoSpritez:
        MOVE.W  d0,$96(A5) ; DISABILITA DMA

        move.l  BaseVBR(PC),a0          ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        MOVE.W  OLDDMAL(PC),$96(A5)     ; Rimetti il vecchio status DMA
        MOVE.W  OLDINTENAL(PC),$9A(A5)  ; INTENA STATUS
        MOVE.W  OLDINTREQL(PC),$9C(A5)  ; INTREQ
        rts

*****************************************************************************
; Questa routine aspetta D1 fotogrammi. Ogni 50 fotogrammi passa 1 secondo.
;
; d1 = numero di fotogrammi da attendere
;
*****************************************************************************

AspettaBlanks:
        LEA     $DFF000,A5      ; CUSTOM REG per OFFSETS
WBLAN1xb:
        MOVE.w  #$80,D0
WBLAN1bxb:
        CMP.B   6(A5),D0        ; vhposr
        BNE.S   WBLAN1bxb
WBLAN2xb:
        CMP.B   6(A5),D0        ; vhposr
        Beq.S   WBLAN2xb
        DBRA    D1,WBLAN1xb
        rts

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
```

```
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero


                ;5432109876543210
        dc.w    $100,%0110101000000000  ; bplcon0 - 320*256 HAM!

        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
        dc.w $1a0,$0666

        dc.w    $9707,$FFFE     ; wait linea $97

        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $b907,$fffe     ; WAIT - attendi linea $b9
BPLPOINTERS2:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "

        dc.w    $100,%0101001000000000  ; BPLCON0 - 5 bitplanes LOWRES

; La palette, che sara' "ruotata" in 2 gruppi di 16 colori.

cols:
        dc.w $180,$040,$182,$050,$184,$060,$186,$080    ; tono verde
        dc.w $188,$090,$18a,$0b0,$18c,$0c0,$18e,$0e0
        dc.w $190,$0f0,$192,$0d0,$194,$0c0,$196,$0a0
        dc.w $198,$090,$19a,$070,$19c,$060,$19e,$040

        dc.w $1a0,$029,$1a2,$02a,$1a4,$13b,$1a6,$24b    ; tono blu
        dc.w $1a8,$35c,$1aa,$36d,$1ac,$57e,$1ae,$68f
        dc.w $1b0,$79f,$1b2,$68f,$1b4,$58e,$1b6,$37e
        dc.w $1b8,$26d,$1ba,$15d,$1bc,$04c,$1be,$04c
cole:

        dc.w    $da07,$fffe     ; WAIT - attendi la linea $da
        dc.w    $100,$200       ; BPLCON0 - disabilita i bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $ff07,$fffe     ; WAIT - attendi la linea $ff
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, per
                                ; suonare la musica (anche mentre stiamo
                                ; caricando con la dos.library).
```

```
        dc.w    $FFFF,$FFFE     ; Fine della copperlist


*****************************************************************************
;               DISEGNO 320*34 a 5 bitplanes (32 colori)
*****************************************************************************

PICTURE2:
        INCBIN  "pic320*34*5.raw"

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        ;incbin "assembler2:sorgenti4/mod.fairlight"

*****************************************************************************
; Buffer dove viene caricata l'immagine da disco (o hard disk) tramite doslib
*****************************************************************************

        section mioplanaccio,bss_C

buffer:

;buffer2:
Music:
        ds.b    2374    ; 6 bitplanes * 176 lines * 40 bytes (HAM)


        end
```

In questo esempio carichiamo il logo, che appare subito sopra. Se lo caricate da dischetto noterete che appare plane per plane, a pezzi, infatti carica un po' alla volta! Sarebbe meglio caricarlo in un buffer a parte, poi farlo visualizzare tutto insieme a caricamento avvenuto.
La cosa fondamentale del caricamento e' che li tempo atteso dopo il load, prima di richiudere tutto, sia sufficiente. Altrimenti e' la fine!
In Interrupt non viene eseguita la routine dei colori, ma solo quella della musica, almeno si nota il tempo che si attende "per sicurezza".
Dato che questo tempo va atteso comunque, sarebbe intelligente fare un fade o qualche routine che perde tempo facendo qualcosa di carino, prima di richiudere tutto, almeno si e' atteso il tempo, ma non senza usarlo!

Lezione11o2b2

```
; Lezione11o2.s   Caricamento di un file dati usando la dos.library
;                 Premere il tasto sinistro per caricare, destro per uscire

        Section DosLoad,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************
```

```
; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

               ;5432109876543210
DMASET  EQU    %1000001110000000       ; copper,bitplane DMA abilitati

WaitDisk       EQU    30       ; 50-150 al salvataggio (secondo i casi)

START:

; Puntiamo la PIC

       MOVE.L  #PICTURE2,d0
       LEA     BPLPOINTERS2,A1
       MOVEQ   #5-1,D1                 ; num di bitplanes
POINTBT2:
       move.w  d0,6(a1)
       swap    d0
       move.w  d0,2(a1)
       swap    d0
       add.l   #34*40,d0               ; lunghezza del bitplane
       addq.w  #8,a1
       dbra    d1,POINTBT2     ; Rifai D1 volte (D1=num do bitplanes)

; Puntiamo la PIC che sara' caricata (ora e' solo un buffer vuoto)

       LEA     bplpointers,A0
       MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
       MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop:
       MOVE.W  D0,6(A0)
       SWAP    D0
       MOVE.W  D0,2(A0)
       SWAP    D0
       ADDQ.w  #8,A0
       ADD.L   #176*40,D0      ; lunghezza plane
       DBRA    D7,pointloop


; Puntiamo il nostro int di livello 3

       move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
       move.l  oldint6c(PC),crappyint  ; Per DOS LOAD - salteremo all'oldint
       move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

       MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                       ; e sprites.
       move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
       move.w  d0,$88(a5)              ; Facciamo partire la COP
       move.w  #0,$1fc(a5)             ; Disattiva l'AGA
       move.w  #$c00,$106(a5)          ; Disattiva l'AGA
       move.w  #$11,$10c(a5)           ; Disattiva l'AGA

       movem.l d0-d7/a0-a6,-(SP)
       bsr.w   mt_init         ; inizializza la routine musicale
       movem.l (SP)+,d0-d7/a0-a6

       move.w  #$c020,$9a(a5)  ; INTENA - abilito interrupt "VERTB"
                               ; del livello 3 ($6c)

mouse:
       btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
```

```
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).

        bsr.w   DosLoad         ; Carica un file legalmente con la dos.lib
                                ; mentre stiamo visualizzando una nostra
                                ; copperlist e eseguendo un nostro interrupt
        TST.L   ErrorFlag
        bne.s   ErroreLoad      ; File non caricato?? Non usiamolo allora!

mouse2:
        btst    #2,$dff016      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse2          ; loop in modo utente, e ogni vertical blank

ErroreLoad:
        bsr.w   mt_end          ; fine del replay!

        rts                     ; esci

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        bsr.w   ColorCicla              ; Cicla i colori della pic
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
NointVERTB:
        move.w  #$70,$dff09c    ; INTREQ - int eseguito, cancello la richiesta
                                ; dato che il 680x0 non la cancella da solo!!!
        rte                     ; Uscita dall'int VERTB

*****************************************************************************
*       Routine che "cicla" i colori di tutta la palette.                   *
*       Questa routine cicla i primi 15 colori separatamente dal secondo    *
*       secondo blocco di colori. Funziona come i "RANGE" del Dpaint.       *
*****************************************************************************

;       Il contatore "cont" serve a far aspettare 3 fotogrammi prima di
;       eseguire la routine cont. In pratica a "rallentare" l'esecuzione

cont:
        dc.w    0

ColorCicla:
        addq.b  #1,cont
        cmp.b   #3,cont         ; Agisci 1 volta ogni 3 fotogrammi solamente
        bne.s   NonAncora       ; Non siamo ancora al terzo? Esci!
        clr.b   cont            ; Siamo al terzo, azzera il contatore

; Roteazione all'indietro dei primi 15 colori

        lea     cols+2,a0       ; Indirizzo primo colore del primo gruppo
        move.w  (a0),d0         ; Salva il primo colore in d0
        moveq   #15-1,d7        ; 15 colori da "roteare" nel primo gruppo
cloop1:
        move.w  4(a0),(a0)      ; Copia il colore avanti in quello prima
        addq.w  #4,a0           ; salta alla prossimo col. da "indietreggiare"
        dbra    d7,cloop1       ; ripeti d7 volte
```

```
        move.w  d0,(a0)         ; Sistema il primo colore salvato come ultimo.

; Roteazione in avanti dei secondi 15 colori

        lea     cole-2,a0       ; Indirizzo ultimo colore del secondo gruppo
        move.w  (a0),d0         ; Salva l'ultimo colore in d0
        moveq   #15-1,d7        ; Altri 15 colori da "roteare" separatamente
cloop2:
        move.w  -4(a0),(a0)     ; Copia il colore indietro in quello dopo
        subq.w  #4,a0           ; salta al precedente col. da "avanzare"
        dbra    d7,cloop2       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema l'ultimo colore salvato come primo
NonAncora:
        rts


*****************************************************************************
; Routine che carica un file mentre stiamo battendo nel metallo.
*****************************************************************************

DosLoad:
        bsr.w   PreparaLoad     ; Rispristina multitask e setta interrupt load

        moveq   #5,d1           ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 5 frames

        bsr.s   CaricaFile      ; Carica il file con la dos.library
        move.l  d0,ErrorFlag    ; Salva lo stato di successo o di errore

; nota: ora dobbiamo attendere che il motore del disk drive, o la spia del
; povero Hard Disk o CD-ROM si spenga, prima di bloccare tutto, o causiamo
; un crash del sistema spettacolare.

        move.w  #150,d1         ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 150 frames

        bsr.w   DopoLoad        ; Disabilita multitask e rimetti interrupt
        rts

ErrorFlag:
        dc.l    0

*****************************************************************************
; Routine che carica un file di una lunghezza specificata e con un nome
; specificato. Occorre mettere l'intero path!
*****************************************************************************

CaricaFile:

        MOVE.L  DosBase(PC),A6
        MOVE.L  #filename,d1    ; Salva il suo handle
        MOVE.L  #1,d2   ; Salva il suo handle
        MOVE.L  #81158,d3       ; Salva il suo handle
        JSR     -$42(A6)        ; SEEK - "Seeka" il file

        move.l  #filename,d1    ; indirizzo con stringa "file name + path"
        MOVE.L  #$3ED,D2        ; AccessMode: MODE_OLDFILE - File che esiste
                                ; gia', e che quindi potremo leggere.
        MOVE.L  DosBase(PC),A6
        JSR     -$1E(A6)        ; LVOOpen - "Apri" il file
        MOVE.L  D0,FileHandle   ; Salva il suo handle
        BEQ.S   ErrorOpen       ; Se d0 = 0 allora c'e' un errore!
```

```
; Carichiamo il file

        MOVE.L  D0,D1           ; FileHandle in d1 per il Read
        MOVE.L  #buffer,D2      ; Indirizzo Destinazione in d2
        MOVE.L  #42240,D3       ; Lunghezza del file (ESATTA!)
        MOVE.L  DosBase(PC),A6
        JSR     -$2A(A6)        ; LVORead - leggi il file e copialo nel buffer
        cmp.l   #-1,d0          ; Errore? (qua e' indicato con -1)
        beq.s   ErroreRead

; Chiudiamolo

        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
        JSR     -$24(A6)        ; LVOClose - chiudi il file.

; Attenzione al fatto che se non CHIUDETE il file, gli altri programmi non
; potranno accedere a tale file (non potrete ne' cancellarlo, ne' spostarlo).

        moveq   #0,d0   ; Segnaliamo il successo
        rts

; Qua ci son le dolenti note, in caso di errore:

ErroreRead:
        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        MOVE.L  DosBase(PC),A6
        JSR     -$24(A6)        ; LVOClose - chiudi il file.
ErrorOpen:
        moveq   #-1,d0  ; segnaliamo l'insuccesso
        rts


FileHandle:
        dc.l    0

; Stringa di testo, da terminare con uno 0, a cui dovra' puntare d1 prima di
; fare l'OPEN della dos.lib. Conviene mettere l'intero path.

Filename:
        dc.b    "assembler2:sorgenti7/amiet.raw",0      ; path+nomefile
        even

*****************************************************************************
; Routine di interrupt da mettere durante il caricamento. Le routines che
; saranno messe in questo interrupt saranno eseguite anche durante il
; caricamento, sia che avvenga da floppy disk, da Hard Disk, o CD ROM.
; DA NOTARE CHE STIAMO USANDO L'INTERRUPT COPER, E NON QUELLO VBLANK,
; QUESTO PERCHE' DURANTE IL CARICAMENTO DA DISCO, SPECIALMENTE SOTTO KICK 1.3,
; L'INTERRUPT VERTB NON E' STABILE, tanto che la musica avrebbe dei sobbalzi.
; Invece, se mettiamo un "$9c,$8010" nella nostra copperlist, siamo sicuri
; che questa routine sara' eseguita una volta sola per fotogramma.
*****************************************************************************

myint6cLoad:
        btst.b  #4,$dff01f      ; INTREQR - il bit 4, COPER, e' azzerato?
        beq.s   nointL          ; Se si, non e' un "vero" int COPER!
        move.w  #%10000,$dff09c ; Se no, e' la volta buona, togliamo il req!
        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_music        ; Suona la musica
        movem.l (SP)+,d0-d7/a0-a6
```

```
nointL:
        dc.w    $4ef9           ; val esadecimale di JMP
Crappyint:
        dc.l    0               ; Indirizzo dove Jumpare, da AUTOMODIFICARE...
                                ; ATTENZIONE: il codice automodificante non
                                ; andrebbe usato. Comunque se si chiama un
                                ; ClearMyCache prima e dopo, funziona!

*****************************************************************************
; Routine che ripristina il sistema operativo, tranne la copperlist, e in
; piu' setta un interrupt $6c nostro, che poi salta a quello di sistema.
; Da notare che durante il load l'interrupt e' gestito dall'int "COPER"
*****************************************************************************

PreparaLoad:
        LEA     $DFF000,A5              ; Base dei registri CUSTOM per Offsets
        MOVE.W  $2(A5),OLDDMAL          ; Salva il vecchio status di DMACON
        MOVE.W  $1C(A5),OLDINTENAL      ; Salva il vecchio status di INTENA
        MOVE.W  $1E(A5),OLDINTREQL      ; Salva il vecchio status di INTREQ
        MOVE.L  #$80008000,d0           ; Prepara la maschera dei bit alti
        OR.L    d0,OLDDMAL              ; Setta il bit 15 dei valori salvati
        OR.W    d0,OLDINTREQL           ; dei registri, per poterli rimettere.

        bsr.w   ClearMyCache

        MOVE.L  #$7FFF7FFF,$9A(a5)      ; DISABILITA GLI INTERRUPTS & INTREQS

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  OldInt64(PC),$64(a0) ; Sys int liv1 salvato (softint,dskblk)
        move.l  OldInt68(PC),$68(a0) ; Sys int liv2 salvato (I/O,ciaa,int2)
        move.l  #myint6cLoad,$6c(a0) ; Int che poi salta a quello di sys.
        move.l  OldInt70(PC),$70(a0) ; Sys int liv4 salvato (audio)
        move.l  OldInt74(PC),$74(a0) ; Sys int liv5 salvato (rbf,dsksync)
        move.l  OldInt78(PC),$78(a0) ; Sys int liv6 salvato (exter,ciab,inten)

        MOVE.W  #%1000001001010000,$96(A5) ; Abilita blit e disk per sicurezza
        MOVE.W  OLDINTENA(PC),$9A(A5)   ; INTENA STATUS
        MOVE.W  OLDINTREQ(PC),$9C(A5)   ; INTREQ
        move.w  #$c010,$9a(a5)          ; dobbiamo essere sicuri che COPER
                                        ; (interrupt via copperlist) sia ON!

        move.l  4.w,a6
        JSR     -$7e(A6)        ; Enable
        JSR     -$8a(a6)        ; Permit

        MOVE.L  GfxBase(PC),A6
        jsr     -$E4(A6)        ; Aspetta la fine di eventuali blittate
        JSR     -$E4(A6)        ; WaitBlit
        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
                                ; (nel kick 1.3 serve per caricare da disk)
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #0,D0           ; Priorita' in d0 (-128, +127) - NORMALE
                                ; (Per permettere ai drives di respirare)
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)
        rts

OLDDMAL:
        dc.w    0
```

```
OLDINTENAL:             ; Vecchio status INTENA
        dc.w    0
OLDINTREQL:             ; Vecchio status INTREQ
        DC.W    0

*****************************************************************************
; Routine che richiude il sistema operativo e rimette il nostro interrupt
*****************************************************************************

DopoLoad:
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #127,D0         ; Priorita' in d0 (-128, +127) - MASSIMA
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)

        JSR     -$84(a6)        ; Forbid
        JSR     -$78(A6)        ; Disable

        MOVE.L  GfxBase(PC),A6
        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.
        jsr     -$E4(A6)        ; WaitBlit - Attende la fine di ogni blittata
        JSR     -$E4(A6)        ; WaitBlit

        bsr.w   ClearMyCache

        LEA     $dff000,a5              ; Custom base per offsets
AspettaF:
        MOVE.L  4(a5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        AND.L   #$1ff00,D0      ; Seleziona solo i bit della pos. verticale
        CMP.L   #$12d00,D0      ; aspetta la linea $12d per evitare che
        BEQ.S   AspettaF        ; spegnendo i DMA si abbiano sfarfallamenti

        MOVE.L  #$7FFF7FFF,$9A(A5)      ; DISABILITA GLI INTERRUPTS & INTREQS

                ; 5432109876543210
        MOVE.W  #%0000010101110000,d0   ; DISABILITA DMA

        btst    #8-8,olddmal    ; test bitplane
        beq.s   NoPlanesA
        bclr.l  #8,d0           ; non spengere planes
NoPlanesA:
        btst    #5,olddmal+2    ; test sprite
        beq.s   NoSpritez
        bclr.l  #5,d0           ; non spengere sprite
NoSpritez:
        MOVE.W  d0,$96(A5) ; DISABILITA DMA

        move.l  BaseVBR(PC),a0          ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        MOVE.W  OLDDMAL(PC),$96(A5)     ; Rimetti il vecchio status DMA
        MOVE.W  OLDINTENAL(PC),$9A(A5)  ; INTENA STATUS
        MOVE.W  OLDINTREQL(PC),$9C(A5)  ; INTREQ
        rts

*****************************************************************************
; Questa routine aspetta D1 fotogrammi. Ogni 50 fotogrammi passa 1 secondo.
;
; d1 = numero di fotogrammi da attendere
;
```

```
*****************************************************************************

AspettaBlanks:
        LEA     $DFF000,A5      ; CUSTOM REG per OFFSETS
WBLAN1xb:
        MOVE.w  #$80,D0
WBLAN1bxb:
        CMP.B   6(A5),D0        ; vhposr
        BNE.S   WBLAN1bxb
WBLAN2xb:
        CMP.B   6(A5),D0        ; vhposr
        Beq.S   WBLAN2xb
        DBRA    D1,WBLAN1xb
        rts

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler2:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero


                ;5432109876543210
        dc.w    $100,%0110101000000000  ; bplcon0 - 320*256 HAM!

        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
        dc.w $1a0,$0666

        dc.w    $9707,$FFFE     ; wait linea $97

        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $b907,$fffe     ; WAIT - attendi linea $b9
BPLPOINTERS2:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
```

```
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "

        dc.w    $100,%0101001000000000  ; BPLCON0 - 5 bitplanes LOWRES

; La palette, che sara' "ruotata" in 2 gruppi di 16 colori.

cols:
        dc.w $180,$040,$182,$050,$184,$060,$186,$080    ; tono verde
        dc.w $188,$090,$18a,$0b0,$18c,$0c0,$18e,$0e0
        dc.w $190,$0f0,$192,$0d0,$194,$0c0,$196,$0a0
        dc.w $198,$090,$19a,$070,$19c,$060,$19e,$040

        dc.w $1a0,$029,$1a2,$02a,$1a4,$13b,$1a6,$24b    ; tono blu
        dc.w $1a8,$35c,$1aa,$36d,$1ac,$57e,$1ae,$68f
        dc.w $1b0,$79f,$1b2,$68f,$1b4,$58e,$1b6,$37e
        dc.w $1b8,$26d,$1ba,$15d,$1bc,$04c,$1be,$04c
cole:

        dc.w    $da07,$fffe     ; WAIT - attendi la linea $da
        dc.w    $100,$200       ; BPLCON0 - disabilita i bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $ff07,$fffe     ; WAIT - attendi la linea $ff
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, per
                                ; suonare la musica (anche mentre stiamo
                                ; caricando con la dos.library).

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


*****************************************************************************
;               DISEGNO 320*34 a 5 bitplanes (32 colori)
*****************************************************************************

PICTURE2:
        INCBIN  "pic320*34*5.raw"

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler2:sorgenti4/mod.fairlight"

*****************************************************************************
; Buffer dove viene caricata l'immagine da disco (o hard disk) tramite doslib
*****************************************************************************

        section mioplanaccio,bss_C

buffer:
LOGO:
        ds.b    6*40*176        ; 6 bitplanes * 176 lines * 40 bytes (HAM)


        end
```

```
In questo esempio carichiamo il logo, che appare subito sopra. Se lo caricate
da dischetto noterete che appare plane per plane, a pezzi, infatti carica
un po' alla volta! Sarebbe meglio caricarlo in un buffer a parte, poi farlo
visualizzare tutto insieme a caricamento avvenuto.
La cosa fondamentale del caricamento e' che li tempo atteso dopo il load,
prima di richiudere tutto, sia sufficiente. Altrimenti e' la fine!
In Interrupt non viene eseguita la routine dei colori, ma solo quella della
musica, almeno si nota il tempo che si attende "per sicurezza".
Dato che questo tempo va atteso comunque, sarebbe intelligente fare un
fade o qualche routine che perde tempo facendo qualcosa di carino, prima di
richiudere tutto, almeno si e' atteso il tempo, ma non senza usarlo!
```

Lezione11o3

```
; Lezione11o3.s   Caricamento di un file dati usando la dos.library
;                 nota: non occorre sapere la lunghezza del file!
;                 Premere il tasto sinistro per caricare, destro per uscire

        Section DosLoad,code

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; salva interrupt, dma eccetera.
*****************************************************************************

; Con DMASET decidiamo quali canali DMA aprire e quali chiudere

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper,bitplane DMA abilitati

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

; Puntiamo la PIC

        MOVE.L  #PICTURE2,d0
        LEA     BPLPOINTERS2,A1
        MOVEQ   #5-1,D1                 ; num di bitplanes
POINTBT2:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #34*40,d0               ; lunghezza del bitplane
        addq.w  #8,a1
        dbra    d1,POINTBT2     ; Rifai D1 volte (D1=num do bitplanes)

; Puntiamo ad un buffer vuoto, che rimarra' sempre vuoto...

        LEA     bplpointers,A0
        MOVE.L  #LOGO+40*40,d0  ; indirizzo logo (un po' ribassato)
        MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
        ADDQ.w  #8,A0
```

```
        ADD.L   #176*40,D0      ; lunghezza plane
        DBRA    D7,pointloop


; Puntiamo il nostro int di livello 3

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  oldint6c(PC),crappyint  ; Per DOS LOAD - salteremo all'oldint
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_init         ; inizializza la routine musicale
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$c020,$9a(a5)  ; INTENA - abilito interrupt "VERTB"
                                ; del livello 3 ($6c)

mouse:
        btst    #6,$bfe001      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse           ; loop in modo utente, e ogni vertical blank
                                ; nonche' ogni WAIT della linea raster $a0
                                ; lo interrompe per suonare la musica!).


        lea     FileName1(PC),a0        ; Nome del file da caricare
        lea     Buffile1(PC),a1         ; In che label salvo l'indirizzo
                                        ; del file caricato in mem
        lea     Grandezz1(PC),a2        ; In che label salvo la grandezza
        moveq   #2,d0                   ; Tipo di mem. destinaz.: CHIP RAM!

        bsr.w   DosLoad2        ; Carica un file legalmente con la dos.lib
                                ; mentre stiamo visualizzando una nostra
                                ; copperlist e eseguendo un nostro interrupt.
                                ; Definire prima "TypeOfMem", con "2" per CHIP
                                ; o "1" per PUBLIC, e gli altri parametri.
        TST.B   d1              ; Abbiamo Allocato memoria da liberare?
        beq.s   NonFreeMemare   ; d1=0, allora non allocato
        st.b    FreeMema1       ; Se si, setta FreeMema1
NonFreeMemare
        TST.L   d0              ; Controlla se ci sono stati errori...
        bne.s   ErroreLoad      ; File non caricato?? Non usiamolo allora!
                                ; d0=0, allora va tutto bene

; Ora puntiamo alla figura caricata:

        LEA     bplpointers,A0
        move.l  Buffile1(PC),d0 ; indirizzo del file caricato
        ADD.L   #40*40,d0       ; logo un po' ribassato (centriamolo...)
        MOVEQ   #6-1,D7         ; 6 bitplanes HAM.
pointloop2:
        MOVE.W  D0,6(A0)
        SWAP    D0
        MOVE.W  D0,2(A0)
        SWAP    D0
```

```
        ADDQ.w  #8,A0
        ADD.L   #176*40,D0      ; lunghezza plane
        DBRA    D7,pointloop2

ErroreLoad:
mouse2:
        btst    #2,$dff016      ; Mouse premuto? (il processore esegue questo
        bne.s   mouse2          ; loop in modo utente, e ogni vertical blank

        bsr.w   mt_end          ; fine del replay!

; RICORDATEVI SEMPRE DI LIBERARE TUTTA LA MEMORIA ALLOCATA!
; Ma anche di non chiamare il FreeMem se non abbiamo veramente allocato
; la memoria, pena super guru meditation - software failure - casini vari.

        tst.b   FreeMema1       ; c'e' stato un errore prima dell'AllocMem?
        beq.s   NonEraAllocata1 ; se si, non fare il FreeMem!

        move.l  Grandezz1(PC),d0 ; Grandezza del blocco in bytes
        move.l  Buffile1(PC),a1 ; Indirizzo del blocco di mem. allocata
        move.l  4.w,a6
        jsr     -$d2(a6)        ; FreeMem
NonEraAllocata1:
        rts                     ; esci


; Variabili che abbiamo usato per salvare grandezza e indirizzo del file da
; usare e deallocare alla fine

Buffile1:
        dc.l    0
Grandezz1:
        dc.l    0
FreeMema1:
        dc.l    0

*****************************************************************************
*       ROUTINE IN INTERRUPT $6c (livello 3) - usato il VERTB e COPER.
*****************************************************************************

MioInt6c:
        btst.b  #5,$dff01f      ; INTREQR - il bit 5, VERTB, e' azzerato?
        beq.s   NointVERTB              ; Se si, non e' un "vero" int VERTB!
        movem.l d0-d7/a0-a6,-(SP)       ; salvo i registri nello stack
        bsr.w   mt_music                ; suono la musica
        bsr.w   ColorCicla              ; Cicla i colori della pic
        movem.l (SP)+,d0-d7/a0-a6       ; riprendo i reg. dallo stack
NointVERTB:
        move.w  #$70,$dff09c    ; INTREQ - int eseguito, cancello la richiesta
                                ; dato che il 680x0 non la cancella da solo!!!
        rte                     ; Uscita dall'int VERTB

*****************************************************************************
*       Routine che "cicla" i colori di tutta la palette.                   *
*       Questa routine cicla i primi 15 colori separatamente dal secondo    *
*       secondo blocco di colori. Funziona come i "RANGE" del Dpaint.       *
*****************************************************************************

;       Il contatore "cont" serve a far aspettare 3 fotogrammi prima di
;       eseguire la routine cont. In pratica a "rallentare" l'esecuzione

cont:
```

```
        dc.w    0

ColorCicla:
        addq.b  #1,cont
        cmp.b   #3,cont         ; Agisci 1 volta ogni 3 fotogrammi solamente
        bne.s   NonAncora       ; Non siamo ancora al terzo? Esci!
        clr.b   cont            ; Siamo al terzo, azzera il contatore

; Roteazione all'indietro dei primi 15 colori

        lea     cols+2,a0       ; Indirizzo primo colore del primo gruppo
        move.w  (a0),d0         ; Salva il primo colore in d0
        moveq   #15-1,d7        ; 15 colori da "roteare" nel primo gruppo
cloop1:
        move.w  4(a0),(a0)      ; Copia il colore avanti in quello prima
        addq.w  #4,a0           ; salta alla prossimo col. da "indietreggiare"
        dbra    d7,cloop1       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema il primo colore salvato come ultimo.

; Roteazione in avanti dei secondi 15 colori

        lea     cole-2,a0       ; Indirizzo ultimo colore del secondo gruppo
        move.w  (a0),d0         ; Salva l'ultimo colore in d0
        moveq   #15-1,d7        ; Altri 15 colori da "roteare" separatamente
cloop2:
        move.w  -4(a0),(a0)     ; Copia il colore indietro in quello dopo
        subq.w  #4,a0           ; salta al precedente col. da "avanzare"
        dbra    d7,cloop2       ; ripeti d7 volte
        move.w  d0,(a0)         ; Sistema l'ultimo colore salvato come primo
NonAncora:
        rts


*****************************************************************************
; Routine che carica un file mentre stiamo battendo nel metallo.
;
; Parametri in entrata:
;
; a0    = Indirizzo della stringa col nome del file da caricare
; a1    = Indirizzo della label (.L) dove salvare l'indirizzo del file
; a2    = Indirizzo della label (.L) dove salvare la lunghezza del file
;
; Parametri in uscita:
;
; d0.l  = Se = 0, non ci sono problemi, se = -1 c'e' stato qualche errore
; d1.b  = Se = 0 non abbiamo eseguito AllocMem, se = 1 dobbiamo fare FreeMem!
;
*****************************************************************************

DosLoad2:
        movem.l d2-d7/a3-a6,-(SP)
        move.l  a0,FileName     ; Nome del file da caricare
        move.l  a1,DestinazLoad ; In che label salvo l'indirizzo
        move.l  a2,SalvGrandezz ; In che label salvo la grandezza file
        move.l  d0,TypeOfMem    ; Tipo di mem. destinaz.: CHIP RAM!

        bsr.w   PreparaLoad     ; Rispristina multitask e setta interrupt load

        moveq   #5,d1           ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 5 frames

        bsr.s   CaricaFile2     ; Carica il file con la dos.library, di
```

```
                                ; qualunque lunghezza "sconosciuta".
        move.l  d0,ErrorFlag    ; Salva lo stato di successo o di errore

; nota: ora dobbiamo attendere che il motore del disk drive, o la spia del
; povero Hard Disk o CD-ROM si spenga, prima di bloccare tutto, o causiamo
; un crash del sistema spettacolare.

        move.w  #150,d1         ; num. di frames da aspettare
        bsr.w   AspettaBlanks   ; aspetta 150 frames

        bsr.w   DopoLoad        ; Disabilita multitask e rimetti interrupt
        move.l  ErrorFlag(PC),d0 ; metti il segnalatore di errore/successo
        moveq   #0,d1
        move.b  FreeMemFlag(PC),d1 ; metti il segnalatore di freemem da fare
        clr.b   FreeMemFlag        ; azzeralo (non allocated) per la prossima
        movem.l (SP)+,d2-d7/a3-a6
        rts

ErrorFlag:
        dc.l    0

*****************************************************************************
; Routine che carica un file di lunghezza "sconosciuta" qualsiasi in un
; buffer di memoria CHIP o PUBLIC, allocato con AllocMem.
; Definire prima "TypeOfMem", con "2" per CHIP RAM o "1" per PUBLIC.
; In uscita, d0=0 se ha avuto successo, oppure -1 se ci sono stati problemi
*****************************************************************************

;        ,,~‘‘
;       ( o o )
;+-.oooO--(_)--Oooo.----------------------------------------------------------+
;|                                                                            |
;|   Oooo               SCUSATE SE LA ROUTINE E’ UN PO’ TROPPO INCASINATA     |
;|   (   )   Oooo.                                                            |
;+----\ (----(   )-----------------------------------------------------------+
;      \_)    ) /
;            (_/

CaricaFile2:
        move.l  filename(PC),d1 ; indirizzo con stringa "file name + path"
        MOVE.L  #$3ED,D2        ; AccessMode: MODE_OLDFILE - File che esiste
                                ; gia’, e che quindi potremo leggere.
        MOVE.L  DosBase(PC),A6
        JSR     -$1E(A6)        ; LVOOpen - "Apri" il file
        MOVE.L  D0,FileHandle   ; Salva il suo handle
        BEQ.W   ErrorOpen       ; Se d0 = 0 allora c’e’ un errore!

; Ora Facciamo il lock del file, per poterci poi fare l’examine

        move.l  filename(pc),d1 ; nome del file
        moveq   #-2,d2          ; AccessMode = ACCESS_READ
        jsr     -$54(a6)        ; lock del file
        move.l  d0,filelock     ; Salviamo il puntatore al lock del file
        beq.w   ErroreLock      ; d0 = 0? Allora errore!

; Allochiamo memoria per il FileInfoBlock di Examine()

        move.l  #$104,d0        ; Grandezza del blocco in bytes
        move.l  #1,d1           ; Tipo di Memoria: public
        move.l  4.w,a6
        jsr     -$c6(a6)        ; Allocmem
        move.l  d0,FibAdr       ; Indirizzo inizio del blocco di mem. allocata
```

```
        beq.s   ErroreAlFib     ; d0=0? Allora errore!

; Ora eseguiamo l'Examine del file, per sapere la sua lunghezza

        move.l  FileLock(PC),d1 ; lock ptr in d1 per examine
        move.l  FibAdr(PC),d2   ; File Info Block per examine
        MOVE.L  DosBase(PC),A6
        jsr     -$66(a6)        ; examine che riempie il buffer fib ($104 byte)
                                ; di infos sul file(nome, dir/file, size, date)
        tst.l   d0              ; Problemi con Examine?
        beq.s   ErroreExamine
        move.l  FibAdr(pc),a0
        move.l  $7c(a0),d3      ; offset del size (lunghezza)
        move.l  d3,SizeFile

; Ora possiamo anche liberare la memoria usata per il FileInfoBlock

        bsr.w   LiberaFib

; Allochiamo la memoria per il file

        move.l  SizeFile(PC),d0  ; Grandezza del blocco in bytes
        move.l  TypeOfMem(PC),d1 ; Tipo di Memoria
        move.l  4.w,a6
        jsr     -$c6(a6)        ; Allocmem
        move.l  d0,FileAdr      ; Indirizzo inizio del blocco di mem. allocata
        beq.s   ErroreAllFile   ; d0=0? Allora errore!

; Facciamo una copia dell'indirizzo del file per noi e il freemem

        move.l  DestinazLoad(PC),a0
        move.l  d0,(a0)
        move.l  SalvGrandezz(PC),a0     ; e salviamo anche la lunghezza
        move.l  SizeFile(PC),(a0)

        st.b    FreeMemFlag     ; Ricordiamoci di fare il freemem

; Carichiamo il file nel blocco di memoria allocato:

        MOVE.L  FileHandle(PC),D1 ; FileHandle in d1 per il Read
        MOVE.L  FileAdr(PC),D2  ; Indirizzo Destinazione in d2
        MOVE.L  SizeFile(PC),D3 ; Lunghezza del file (ESATTA!)
        MOVE.L  DosBase(PC),A6
        JSR     -$2A(A6)        ; LVORead - leggi il file e copialo nel buffer
        cmp.l   #-1,d0          ; Errore? (qua e' indicato con -1)
        beq.s   ErroreRead

; Unlockiamo il file

        bsr.s   UnlockFile

; E chiudiamolo

        bsr.s   CloseFile

; Attenzione al fatto che se non CHIUDETE il file, gli altri programmi non
; potranno accedere a tale file (non potrete ne' cancellarlo, ne' spostarlo).

        moveq   #0,d0   ; segnala il successo totale!
        rts
```

```
; Ecco la compilation degli errori possibili:


ErroreExamine:
        bsr.s   LiberaFib
ErroreAllFile:
ErroreAlFib:
        bsr.s   UnlockFile
ErroreLock:
        bsr.s   CloseFile
ErrorOpen:
        moveq   #-1,d0          ; segnala l'errore
        rts

ErroreRead:
        bsr.s   UnlockFile
        bsr.s   CloseFile
        st.b    FreeMemFlag     ; Ricordiamoci di fare il freemem
        moveq   #-1,d0          ; Segnala l'errore
        rts

; Routines chiamate da piu' parti:

CloseFile:
        MOVE.L  DosBase(PC),A6
        MOVE.L  FileHandle(pc),D1 ; FileHandle in d1
        JSR     -$24(A6)        ; LVOClose - chiudi il file.
        rts

UnlockFile:
        MOVE.L  DosBase(PC),A6
        move.l  FileLock(PC),d1 ; lock ptr in d1 per unlock
        jsr     -$5a(a6)        ; Unlock del file
        rts

LiberaFib:
        move.l  4.w,a6
        move.l  #$104,d0        ; Grandezza del blocco in bytes
        move.l  FibAdr(PC),a1   ; Indirizzo del blocco di mem. allocata
        jsr     -$d2(a6)        ; FreeMem
        rts

SalvGrandezz:
        dc.l    0
DestinazLoad:
        dc.l    0
FileHandle:
        dc.l    0
TypeOfMem:
        dc.l    0
FileName:
        dc.l    0
FileLock:
        dc.l    0
FibAdr:
        dc.l    0
SizeFile:
        dc.l    0
FileAdr
        dc.l    0
FreeMemFlag:
        dc.l    0
```

```
; Stringa di testo, da terminare con uno 0, a cui dovra' puntare d1 prima di
; fare l'OPEN della dos.lib. Conviene mettere l'intero path.


Filename1:
        dc.b    "assembler3:sorgenti7/amiet.raw",0      ; path+nomefile
        even

*****************************************************************************
; Routine di interrupt da mettere durante il caricamento. Le routines che
; saranno messe in questo interrupt saranno eseguite anche durante il
; caricamento, sia che avvenga da floppy disk, da Hard Disk, o CD ROM.
; DA NOTARE CHE STIAMO USANDO L'INTERRUPT COPER, E NON QUELLO VBLANK,
; QUESTO PERCHE' DURANTE IL CARICAMENTO DA DISCO, SPECIALMENTE SOTTO KICK 1.3,
; L'INTERRUPT VERTB NON E' STABILE, tanto che la musica avrebbe dei sobbalzi.
; Invece, se mettiamo un "$9c,$8010" nella nostra copperlist, siamo sicuri
; che questa routine sara' eseguita una volta sola per fotogramma.
*****************************************************************************

myint6cLoad:
        btst.b  #4,$dff01f      ; INTREQR - il bit 4, COPER, e' azzerato?
        beq.s   nointL          ; Se si, non e' un "vero" int COPER!
        move.w  #%10000,$dff09c ; Se no, e' la volta buona, togliamo il req!
        movem.l d0-d7/a0-a6,-(SP)
        bsr.w   mt_music        ; Suona la musica
        movem.l (SP)+,d0-d7/a0-a6
nointL:
        dc.w    $4ef9           ; val esadecimale di JMP
Crappyint:
        dc.l    0               ; Indirizzo dove Jumpare, da AUTOMODIFICARE...
                                ; ATTENZIONE: il codice automodificante non
                                ; andrebbe usato. Comunque se si chiama un
                                ; ClearMyCache prima e dopo, funziona!

*****************************************************************************
; Routine che ripristina il sistema operativo, tranne la copperlist, e in
; piu' setta un interrupt $6c nostro, che poi salta a quello di sistema.
; Da notare che durante il load l'interrupt e' gestito dall'int "COPER"
*****************************************************************************

PreparaLoad:
        LEA     $DFF000,A5              ; Base dei registri CUSTOM per Offsets
        MOVE.W  $2(A5),OLDDMAL          ; Salva il vecchio status di DMACON
        MOVE.W  $1C(A5),OLDINTENAL      ; Salva il vecchio status di INTENA
        MOVE.W  $1E(A5),OLDINTREQL      ; Salva il vecchio status di INTREQ
        MOVE.L  #$80008000,d0           ; Prepara la maschera dei bit alti
        OR.L    d0,OLDDMAL              ; Setta il bit 15 dei valori salvati
        OR.W    d0,OLDINTREQL           ; dei registri, per poterli rimettere.

        bsr.w   ClearMyCache

        MOVE.L  #$7FFF7FFF,$9A(a5)      ; DISABILITA GLI INTERRUPTS & INTREQS

        move.l  BaseVBR(PC),a0       ; In a0 il valore del VBR
        move.l  OldInt64(PC),$64(a0) ; Sys int liv1 salvato (softint,dskblk)
        move.l  OldInt68(PC),$68(a0) ; Sys int liv2 salvato (I/O,ciaa,int2)
        move.l  #myint6cLoad,$6c(a0) ; Int che poi salta a quello di sys.
        move.l  OldInt70(PC),$70(a0) ; Sys int liv4 salvato (audio)
        move.l  OldInt74(PC),$74(a0) ; Sys int liv5 salvato (rbf,dsksync)
        move.l  OldInt78(PC),$78(a0) ; Sys int liv6 salvato (exter,ciab,inten)
```

```
        MOVE.W  #%1000001001010000,$96(A5) ; Abilita blit e disk per sicurezza
        MOVE.W  OLDINTENA(PC),$9A(A5)   ; INTENA STATUS
        MOVE.W  OLDINTREQ(PC),$9C(A5)   ; INTREQ
        move.w  #$c010,$9a(a5)          ; dobbiamo essere sicuri che COPER
                                        ; (interrupt via copperlist) sia ON!

        move.l  4.w,a6
        JSR     -$7e(A6)        ; Enable
        JSR     -$8a(a6)        ; Permit

        MOVE.L  GfxBase(PC),A6
        jsr     -$E4(A6)        ; Aspetta la fine di eventuali blittate
        JSR     -$E4(A6)        ; WaitBlit
        jsr     -$1ce(a6)       ; DisOwnBlitter, il sistema operativo ora
                                ; puo' nuovamente usare il blitter
                                ; (nel kick 1.3 serve per caricare da disk)
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #0,D0           ; Priorita' in d0 (-128, +127) - NORMALE
                                ; (Per permettere ai drives di respirare)
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)
        rts

OLDDMAL:
        dc.w    0
OLDINTENAL:             ; Vecchio status INTENA
        dc.w    0
OLDINTREQL:             ; Vecchio status INTREQ
        DC.W    0

*****************************************************************************
; Routine che richiude il sistema operativo e rimette il nostro interrupt
*****************************************************************************

DopoLoad:
        MOVE.L  4.w,A6
        SUBA.L  A1,A1           ; NULL task - trova questo task
        JSR     -$126(A6)       ; findtask (Task(name) in a1, -> d0=task)
        MOVE.L  D0,A1           ; Task in a1
        MOVEQ   #127,D0         ; Priorita' in d0 (-128, +127) - MASSIMA
        JSR     -$12C(A6)       ;_LVOSetTaskPri (d0=priorita', a1=task)

        JSR     -$84(a6)        ; Forbid
        JSR     -$78(A6)        ; Disable

        MOVE.L  GfxBase(PC),A6
        jsr     -$1c8(a6)       ; OwnBlitter, che ci da l'esclusiva sul blitter
                                ; impedendone l'uso al sistema operativo.
        jsr     -$E4(A6)        ; WaitBlit - Attende la fine di ogni blittata
        JSR     -$E4(A6)        ; WaitBlit

        bsr.w   ClearMyCache

        LEA     $dff000,a5              ; Custom base per offsets
AspettaF:
        MOVE.L  4(a5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        AND.L   #$1ff00,D0      ; Seleziona solo i bit della pos. verticale
        CMP.L   #$12d00,D0      ; aspetta la linea $12d per evitare che
        BEQ.S   AspettaF        ; spegnendo i DMA si abbiano sfarfallamenti
```

```
        MOVE.L  #$7FFF7FFF,$9A(A5)      ; DISABILITA GLI INTERRUPTS & INTREQS

                ; 5432109876543210
        MOVE.W  #%0000010101110000,d0   ; DISABILITA DMA

        btst    #8-8,olddmal    ; test bitplane
        beq.s   NoPlanesA
        bclr.l  #8,d0           ; non spengere planes
NoPlanesA:
        btst    #5,olddmal+2    ; test sprite
        beq.s   NoSpritez
        bclr.l  #5,d0           ; non spengere sprite
NoSpritez:
        MOVE.W  d0,$96(A5) ; DISABILITA DMA

        move.l  BaseVBR(PC),a0          ; In a0 il valore del VBR
        move.l  #MioInt6c,$6c(a0)       ; metto la mia rout. int. livello 3.
        MOVE.W  OLDDMAL(PC),$96(A5)     ; Rimetti il vecchio status DMA
        MOVE.W  OLDINTENAL(PC),$9A(A5)  ; INTENA STATUS
        MOVE.W  OLDINTREQL(PC),$9C(A5)  ; INTREQ
        rts

*****************************************************************************
; Questa routine aspetta D1 fotogrammi. Ogni 50 fotogrammi passa 1 secondo.
;
; d1 = numero di fotogrammi da attendere
;
*****************************************************************************

AspettaBlanks:
        LEA     $DFF000,A5      ; CUSTOM REG per OFFSETS
WBLAN1xb:
        MOVE.w  #$80,D0
WBLAN1bxb:
        CMP.B   6(A5),D0        ; vhposr
        BNE.S   WBLAN1bxb
WBLAN2xb:
        CMP.B   6(A5),D0        ; vhposr
        Beq.S   WBLAN2xb
        DBRA    D1,WBLAN1xb
        rts

*****************************************************************************
;       Routine di replay del protracker/soundtracker/noisetracker
;
        include "assembler3:sorgenti4/music.s"
*****************************************************************************

        SECTION GRAPHIC,DATA_C

COPPERLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

BPLPOINTERS:
        dc.w $e0,0,$e2,0                ;primo   bitplane
```

```
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "
        dc.w $f4,0,$f6,0                ;sesto      "

        dc.w    $180,0  ; Color0 nero


                ;5432109876543210
        dc.w    $100,%0110101000000000  ; bplcon0 - 320*256 HAM!

        dc.w $180,$0000,$182,$134,$184,$531,$186,$443
        dc.w $188,$0455,$18a,$664,$18c,$466,$18e,$973
        dc.w $190,$0677,$192,$886,$194,$898,$196,$a96
        dc.w $198,$0ca6,$19a,$9a9,$19c,$bb9,$19e,$dc9
        dc.w $1a0,$0666

        dc.w    $9707,$FFFE     ; wait linea $97

        dc.w    $100,$200       ; BPLCON0 - no bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $b907,$fffe     ; WAIT - attendi linea $b9
BPLPOINTERS2:
        dc.w $e0,0,$e2,0                ;primo   bitplane
        dc.w $e4,0,$e6,0                ;secondo    "
        dc.w $e8,0,$ea,0                ;terzo      "
        dc.w $ec,0,$ee,0                ;quarto     "
        dc.w $f0,0,$f2,0                ;quinto     "

        dc.w    $100,%0101001000000000  ; BPLCON0 - 5 bitplanes LOWRES

; La palette, che sara' "ruotata" in 2 gruppi di 16 colori.

cols:
        dc.w $180,$040,$182,$050,$184,$060,$186,$080    ; tono verde
        dc.w $188,$090,$18a,$0b0,$18c,$0c0,$18e,$0e0
        dc.w $190,$0f0,$192,$0d0,$194,$0c0,$196,$0a0
        dc.w $198,$090,$19a,$070,$19c,$060,$19e,$040

        dc.w $1a0,$029,$1a2,$02a,$1a4,$13b,$1a6,$24b    ; tono blu
        dc.w $1a8,$35c,$1aa,$36d,$1ac,$57e,$1ae,$68f
        dc.w $1b0,$79f,$1b2,$68f,$1b4,$58e,$1b6,$37e
        dc.w $1b8,$26d,$1ba,$15d,$1bc,$04c,$1be,$04c
cole:

        dc.w    $da07,$fffe     ; WAIT - attendi la linea $da
        dc.w    $100,$200       ; BPLCON0 - disabilita i bitplanes
        dc.w    $180,$00e       ; color0 BLU

        dc.w    $ff07,$fffe     ; WAIT - attendi la linea $ff
        dc.w    $9c,$8010       ; INTREQ - Richiedo un interrupt COPER, per
                                ; suonare la musica (anche mentre stiamo
                                ; caricando con la dos.library).

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


*****************************************************************************
;               DISEGNO 320*34 a 5 bitplanes (32 colori)
*****************************************************************************
```

```
PICTURE2:
        INCBIN  "pic320*34*5.raw"

*****************************************************************************
;                               MUSICA
*****************************************************************************

mt_data:
        dc.l    mt_data1

mt_data1:
        incbin  "assembler3:sorgenti7/Mod.Prova"

*****************************************************************************
; Buffer che rimane vuoto, qua serve solo come "nero" prima del caricamento.
; Avremmo potuto anche semplicemente disabilitare i bitplanes, ed attivarli
; solo a figura caricata, per risparmiare questo spazio azzerato!
*****************************************************************************

        section mioplanaccio,bss_C

buffer:
LOGO:
        ds.b    6*40*176        ; 6 bitplanes * 176 lines * 40 bytes (HAM)


        end
```

Questo listato locka il file, lo EXAMINA per averne la lunghezza, alloca la memoria con AllocMem, lo carica, e alla fine libera il blocco di memoria con FreeMem. Vi prego di notare l'attenzione posta alle varie possibilita' di errore, per non eseguire FreeMem quando non si e' allocato niente, e per non usare il file quando non lo si e' caricato bene. Occorre SEMPRE controllare tutto, e preparare le routine ad eventuali insuccessi.
La routine e' stata resa piu' parametrica, infatti si puo' specificare ogni volta il nome del file e il tipo di memoria richiesto. Basta che in caso voleste caricare piu' files, non vi confondiate con i vari FreeMem alla fine! Ecco un esempio in cui carichiamo 2 files:

```
; Primo file:

        lea     FileName1(PC),a0        ; Nome del file da caricare
        lea     Buffile1(PC),a1         ; In che label salvo l'indirizzo
                                        ; del file caricato in mem
        lea     Grandezz1(PC),a2        ; In che label salvo la grandezza
        moveq   #2,d0                   ; Tipo di mem. destinaz.: CHIP RAM!

        bsr.w   DosLoad2        ; Carica un file legalmente con la dos.lib
                                ; mentre stiamo visualizzando una nostra
                                ; copperlist e eseguendo un nostro interrupt
                                ; Definire prima "TypeOfMem", con "2" per CHIP
                                ; o "1" per PUBLIC, e gli altri parametri.
        TST.B   d1              ; Abbiamo Allocato memoria da liberare?
        beq.s   NonFreeMemare   ; d1=0, allora non allocato
        st.b    FreeMema1       ; Se si, setta FreeMema1
NonFreeMemare
        TST.L   d0              ; Controlla se ci sono stati errori...
        bne.s   ErroreLoad      ; File non caricato?? Non usiamolo allora!
                                ; d0=0, allora va tutto bene
        ....
```

```
; Secondo file:

        lea     FileName2(PC),a0        ; Nome del file da caricare
        lea     Buffile2(PC),a1         ; In che label salvo l'indirizzo
                                        ; del file caricato in mem
        lea     Grandezz2(PC),a2        ; In che label salvo la grandezza
        moveq   #2,d0                   ; Tipo di mem. destinaz.: CHIP RAM!

        bsr.w   DosLoad2        ; Carica un file legalmente con la dos.lib
                                ; mentre stiamo visualizzando una nostra
                                ; copperlist e eseguendo un nostro interrupt
                                ; Definire prima "TypeOfMem", con "2" per CHIP
                                ; o "1" per PUBLIC, e gli altri parametri.
        TST.B   d1              ; Abbiamo Allocato memoria da liberare?
        beq.s   NonFreeMemare2  ; d1=0, allora non allocato
        st.b    FreeMema2       ; Se si, setta FreeMema1
NonFreeMemare2
        TST.L   d0              ; Controlla se ci sono stati errori...
        bne.s   ErroreLoad2     ; File non caricato?? Non usiamolo allora!
        ...                     ; d0=0, allora va tutto bene


E alla fine, dovremo fare 2 FreeMem:

        tst.b   FreeMema1       ; c'e' stato un errore prima dell'AllocMem?
        beq.s   NonEraAllocata1 ; se si, non fare il FreeMem!

        move.l  Grandezz1(PC),d0 ; Grandezza del blocco in bytes
        move.l  Buffile1(PC),a1 ; Indirizzo del blocco di mem. allocata
        move.l  4.w,a6
        jsr     -$d2(a6)        ; FreeMem
NonEraAllocata1:

        tst.b   FreeMema2       ; c'e' stato un errore prima dell'AllocMem?
        beq.s   NonEraAllocata2 ; se si, non fare il FreeMem!

        move.l  Grandezz2(PC),d0 ; Grandezza del blocco in bytes
        move.l  Buffile2(PC),a1 ; Indirizzo del blocco di mem. allocata
        move.l  4.w,a6
        jsr     -$d2(a6)        ; FreeMem
NonEraAllocata2:

        rts                     ; esci


Naturalmente potete cambiare la routine come volete, o rendere parametrica
quella che carica files con lunghezza specificata... fate come vi pare, ma
controllate bene di non fare errori!
```

CAPITOLO 28

LEZIONE 14

28.1 Lezione14-1

```
; Lezione14-1.s                 ** SUONARE UN'ARMONICA **


        SECTION armonica,CODE

Start:
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable

        bset    #1,$bfe001              ; Spegne il filtro passa-basso

        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS

Clock   equ     3546895

        move.l  #armonica,$a0(a6)       ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.w  #16/2,$a4(a6)           ; 16 bytes/2=8 word di dati (AUD0LEN)
        move.w  #clock/(16*880),$a6(a6) ; AUD0PER a 251
        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #$8201,$96(a6)          ; Accende AUD0 DMA in DMACONW

WLMB:
        btst    #6,$bfe001              ; Aspetta tasto sinistro del mouse
        bne.s   WLMB

        or.w    #$8000,d7               ; Accende il bit 15 (SET/CLR)
        move.w  #$0001,$96(a6)          ; dmacon - Spegne aud0
        move.w  d7,$96(a6)              ; dmacon - Reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts
```

```
*****************************************************************************

        SECTION Sample,DATA_C   ;venendo letta dal DMA deve essere in CHIP

        ; Armonica di 16 valori creata col'IS del trash'm-one

Armonica:
        DC.B    $19,$46,$69,$7C,$7D,$6A,$47,$1A,$E8,$BB,$97,$84,$83,$95,$B8,$E5

        END

*****************************************************************************

L'Armonica e' un sample di 16 byte che viene suonato sul canale 0 con periodo
di campionamento 251.
Per suonare 16 byte in 1 secondo (1 Hz), il valore di AUDPER dovrebbe essere
di 1/16 del valore della costante di clock, poiche' il DMA dovrebbe attendere
1/16 del clock per 16 = tutto il clock = 1 secondo.
Per generare un LA3, ad esempio, (=440 Hz) bisognerebbe campionare a 880 Hz per
il teorema di Nyquist, per cui l'armonica andrebbe letta con una frequenza di
lettura di 880 Hz, ed il periodo di campionamento (=valore da inserire in
AUDPER) sarebbe di 1/880 dell' 1/16 della costante di clock:
3546895/16 = 221680 = 1 Hz, valore, oltretutto, non inseribile nel registro
poiche' e' superiore al range dei 16 bit (AUDxPER = 1 word senza segno);
(3546895/16)/880 = 3546895/(16*880) = 251 = 880 Hz.

N.B.:   i due JSR alle funzioni "Disable" ed "Enable" dell'exec potrebbero
        essere omessi, ma, per eleganza di coding, sarebbero obbligatori:
        sotto sistema operativo non sarebbe possibile toccare direttamente
        i canali DMA (nemmeno quelli audio), non tanto per il rischio che vada
        in crash il computer (l'exec non e' ingrado di controllare evetuali
        accessi ai registri hardware, visto che l'hardware non ha circuiti di
        protezione e le librerie di sistema non fanno miracoli), quanto per
        la certezza che il vostro task/processo andra' in conflitto con altri
        task/processi che stanno utilizzando le risorse audio: l'amiga ha solo
        un chip sonoro e tutti devono accedere a quello, per suonare; il
        Kernel in ROM mette a disposizione l'AUDIO.DEVICE per permettere
        a qualunque task di usufruire del chip e per arbitrarne via software
        l'accesso e l'uso tra i vari processi.
        Siccome questo corso prevede l'utilizzo dell'hardware tramite accesso
        diretto ai registri, noi non useremo le device, e, pertanto, saremo
        sempre obbligati (anche se, nel caso in cui nessuno stia accedendo
        all'hardware sonoro, non sarebbe effettivamente necessario) a spegnere
        "legalemente" (con una funzione dell'exec) il sistema operativo.
```

28.2 Lezione14-2a

```
; Lezione14-2a.s        ** SUONARE UN'ARMONICA A VARIE NOTE **


        SECTION Armonica2,CODE

Start:
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable

        bset    #1,$bfe001              ; Spegne il filtro passa-basso

        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS
```

```
Clock   equ     3546895

        move.l  #armonica,$a0(a6)       ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.w  #16/2,$a4(a6)           ; 16 bytes/2=8 word di dati (AUD0LEN)
        move.l  #clock/16,d1            ; 1/16 = un 16esimo del clock
        divu.w  do3(pc),d1              ; <<< CAMBIATE IL PRIMO OPERANDO DI
                                        ;  QUESTO MOVE PER GENERARE ALTRE
                                        ; NOTE >>>
        move.w  d1,$a6(a6)              ; AUD0PER col periodo calcolato
        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #$8001,$96(a6)          ; Accende AUD0 DMA in DMACONW

WLMB:   btst    #6,$bfe001              ; Aspetta il tasto sinistro del mouse
        bne.s   WLMB

        or.w    #$8000,d7               ; accende il bit 15 (SET/CLR)
        move.w  #$0001,$96(a6)          ; dmacon - spegne aud0
        move.w  d7,$96(a6)              ; dmacon - reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts


DO3:    dc.w    528             ;frequenze delle note
RE3:    dc.w    528*9/8
MI3:    dc.w    528*5/4
FA3:    dc.w    528*4/3
SOL3:   dc.w    528*3/2
LA3:    dc.w    528*5/3
SI3:    dc.w    528*15/8
DO4:    dc.w    528*2


*****************************************************************************

        SECTION Sample,DATA_C   ;venendo letta dal DMA deve essere in CHIP

        ; Armonica di 16 valori creata col'IS del trash'm-one

Armonica:
        DC.B    $19,$46,$69,$7C,$7D,$6A,$47,$1A,$E8,$BB,$97,$84,$83,$95,$B8,$E5

        END

*****************************************************************************

Al periodo 1/16 del clock (= 35468095/16) si leggerebbe l'armonica ad 1 Hz,
poiche' e' lunga 16 byte, e - come dicevamo nel primo sorgente -, leggendone
16 al secondo, si legge tutta l'armonica 1 volta al secondo (= 1 Hz, appunto);
dividendo il periodo 1/16 per la frequenza della nota da suonare contenuta
in RAM alla relativa label, si moltiplica la frequenza di lettura di 1 Hz
per la frequenza della nota, appunto, facendo leggere all'hardware l'intera
armonica piu' volte al secondo.

Sarebbe stato possibile raggiungere il medesimo risultato da imserire in
AUD0PER anche con il seguende codice:

        [...]
        move.l  #clock,d1               ; costante di clock
        move.w  do3(pc),d2              ; ...o qualsiasi altra frequenza...
        mulu.w  #16,d2                  ; d2.l = 16*frequenza della nota
```

```
        divu.w  d2,d1                   ; d1.w = clock/(16*freq)
        move.w  d1,$a6(a6)              ; imposta AUD0PER
        [...]
```

Lezione14-2b

```
; Lezione14-2b.s        ** SUONARE UN'ARMONICA A VARIE NOTE 2 **


        SECTION Armonica2b,CODE

Start:
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable
        bset    #1,$bfe001              ; Spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS

        move.l  #armonica,$a0(a6)       ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.w  #16/2,$a4(a6)           ; 16 bytes/2=8 word di dati (AUD0LEN)

        move.l  #1<<16!2,d0             ; suona un RE3
        moveq   #16,d1                  ; lunghezza=16 byte
        bsr.s   note2per
        move.w  d0,$a6(a6)              ; AUD0PER col risultato

        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #$8201,$96(a6)          ; accende AUD0 DMA in DMACONW

WLMB:   btst    #6,$bfe001              ; aspetta il tasto sinistro del mouse
        bne.s   WLMB

        or.w    #$8000,d7               ; accende il bit 15 (SET/CLR)
        move.w  #$0001,$96(a6)          ; dmacon - spegne aud0
        move.w  d7,$96(a6)              ; dmacon - reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts

*****************************************************************************
;                       « Note To Period »
;
; Calcola il periodo da inserire in AUDxPER data la nota e l'ottava
;
; d0hi.w  = nota (0[DO]..6[SI])
; d0lo.w  = ottava (0[1]..3[4])
; d1.w    = lunghezza armonica (in byte)
*****************************************************************************

Clock   equ     3546895
DO1     equ     131                     ; frequenza [Hz] del DO 1a ottava

Note2Per:
        move.w  #do1,d2                 ; d2.w=DO1
        lsl.w   d0,d2                   ; d2.w=DOx (secondo l'ottava in d0lo.w)
        swap    d0                      ; d0lo.w=d0hi.w
        add.w   d0,d0                   ; d0.w=d0.w*4
        add.w   d0,d0                   ; per offset di longword da NOTES
        mulu.w  notes(pc,d0.w),d2       ; d2.l=DOx*num
        divu.w  notes+2(pc,d0.w),d2     ; d2.l=DOx*num/den=frequenza nota
```

```
        mulu.w  d1,d2                   ; d2.l=freq. nota*lunghezza=freq. camp.
        move.l  #clock,d0               ; d0.l=costante di clock
        divu.w  d2,d0                   ; d0.w=clock/freq. camp.
        rts                             ; [d0.w=periodo di campionamento]

Notes:
DO:     dc.w    1,1
RE:     dc.w    9,8
MI:     dc.w    5,4
FA:     dc.w    4,3
SOL:    dc.w    3,2
LA:     dc.w    5,3
SI:     dc.w    15,8


*****************************************************************************

        SECTION Sample,DATA_C   ;venendo letta dal DMA deve essere in CHIP

        ; Armonica di 16 valori creata col'IS del trash'm-one

Armonica:
        DC.B    $19,$46,$69,$7C,$7D,$6A,$47,$1A,$E8,$BB,$97,$84,$83,$95,$B8,$E5

        END

*****************************************************************************
```

Come spiegato nel testo della lezione, ad ogni ottava c'e' un RADDOPPIAMENTO di frequenza, quindi, se il DO della prima ottava ha 131 Hz, il DO2 ha 262 Hz, il DO3 524 Hz, ecc; all'interno della scala ci sono dei rapporti ben precisi tra le frequenze delle 7 note: DO=1, RE=9/8, MI=5/4, FA=4/3, SOL=3/2, LA=5/3, SI=15/8 (ed il DO successivo =2); avendo questa tabella e' assai semplice calcolarsi la frequenza di una nota qualsiasi di un'ottava qualsiasi partendo da una data nota di una data ottava.

La subroutine "Note2Per" vuole, come parametri in entrata: d0 con la nota (da 0 per il DO al 6 per il SI) sulla word alta, e l'ottava (da 0 per la prima a 3 per la quarta) sulla bassa, e d1, con la lunghezza in byte del sample. Essa calcola il periodo di campionamento da inserire nei registri AUDxPER solamente sapendo la frequenza di un DO1 e la nota desiderata.
Come funziona ? Semplice: innanzitutto, notate che i rapporti tra le note rispetto al DO di ogni scala sono stati inseriti come dati di word, con la prima che indica il numeratore della frazione e la seconda che ne indica il denominatore; la routine, per prima cosa, raddoppia la frequenza del DO1 di partenza tante volte quante sono le ottave espresse nella word bassa del parametro d0, semplicemente shiftando a sinistra (moltiplicando *2 ad ogni singolo shift) il valore 132 (frequenza del DO1) di tanti bit quanto il valore in d0lo.w; poi, in base alla nota specificata in d0hi.w, preleva da Notes+d0hi.w*4 un longword avente nella word alta il numeratore ed in quella bassa il denominatore della frazione che indica il rapporto della nota desiderata all'interno della scala dal DO relativo (DO=1/1); in seguito, calcola la frequenza della nota dell'ottava voluta moltiplicando la frazione per la frequenza del DO dell'ottava stessa, e, quindi, moltiplicando questa per il numeratore della frazione (word alta) e dividendo poi il tutto per il denominatore (word bassa); infine, ottenuta la frequenza esatta, viene calcolato il periodo di campionamento affinche' il sample di lunghezza d1.w venga letto INTERAMENTE alla frequenza della nota (come si diceva negli esempi precedenti, non calcoliamo il periodo di campionamento in lettura derivato dal numero di byte al secondo che devono essere letti, ma il periodo di campionamento derivato dal numero di volte che TUTTA l'armonica deve venir letta in 1 secondo, pari al prodotto tra la frequenza e la lunghezza in byte

```
dell'onda: un valore ASSAI maggiore !), dividendo la costante di clock per
il prodotto tra la lunghezza del sample e la frequenza della nota.

Se, per esempio, vogliamo suonare un SOL2, dobbiamo fornire il valore 4 (5a
nota) nella word alta di d0 ed 1 (2a ottava) in quella bassa.
La frequenza della nota sara':
                              ((132 * 2^ottava) * 3)/2
                                 |               /  \
                                DO1            num  den
                                \____________/
                                       |
                                      DOx

** In sostanza, la ruotine prima calcola il DO dell'ottava giusta, poi ne
calcola i 3/2 ("tre mezzi") **.

N.B.:   cosi' come'e', la routine "Note2Per" ha una limitazione: come dovreste
        sapere il 68000 effettua moltiplicazioni 16bit*16bit=32bit e divisioni
        32bit/16bit=16bit (il resto sulla word alta del risultato), percio'
        non e' possibile suonare sample troppo lunghi ad una frequenza troppo
        alta, semplicemente perche' il prodotto tra lunghezza e frequenza deve
        essere diviso per la costante di clock, e quindi, il divisore del
        DIVU deve stare in una word (senza segno, per fortuna).
        ** In pratica, pero', questa limitazione non danneggia nessuno, infatti
        la velocita' di lettura = freq della nota * lunghezza del sample non
        puo' superare i 28836 Hz, valore che sta' comodamente dentro un word:
        NON USATE DUNQUE FREQUENZE TROPPO ALTE PER SAMPLE TROPPO LUNGHI **
```

28.3 Lezione14-3a

```
; Lezione14-3a.s        ** TONI E SEMITONI DI PRECISIONE **


        SECTION Toni,CODE

Start:
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable
        bset    #1,$bfe001              ; Spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS

        move.l  #armonica,$a0(a6)       ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.w  #16/2,$a4(a6)           ; 16 bytes/2=8 word di dati (AUD0LEN)

        move.l  #12*2+2,d0              ; RE3
        moveq   #16,d1
        bsr.s   halftone2per
        move.w  d0,$a6(a6)              ; AUD0PER

        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #$8001,$96(a6)          ; accende AUD0 DMA in DMACONW

WLMB:
        btst    #6,$bfe001              ;aspetta il tasto sinistro del mouse
        bne.s   WLMB

        or.w    #$8000,d7               ; accende il bit 15 (SET/CLR)
        move.w  #$0001,$96(a6)          ; spegne il DMA
        move.w  d7,$96(a6)              ; dmacon - reimposta DMA dell'OS
```

```
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts

*****************************************************************************
;                       « HalfTone To Period »
;
; Calcola il periodo da inserire in AUDxPER data il semitono a partire dal DO1
;
; d0.w   = semitono (a partire dal DO1=0)
; d1.w   = lunghezza armonica (in byte)
*****************************************************************************

Clock   equ     3546895
DO1     equ     131                     ; Frequenza [Hz] del DO 1a ottava

HalfTone2Per:
        divu.w  #12,d0
        move.w  #do1,d2
        lsl.w   d0,d2
        swap    d0
        add.w   d0,d0
        add.w   d0,d0
        mulu.w  halftones(pc,d0.w),d2
        divu.w  halftones+2(pc,d0.w),d2
        move.l  #clock,d0
        mulu.w  d2,d1
        divu.w  d1,d0
        rts                     ; [d0.w=periodo di campionamento]

HalfTones:
        dc.w    10000,10000     ;DO=1.0
        dc.w    10595,10000     ;DO#=1.0595
        dc.w    11225,10000     ;RE=1.1225
        dc.w    11892,10000     ;RE#=1.1892
        dc.w    12599,10000     ;MI=1.2599
        dc.w    13348,10000     ;FA=1.3348
        dc.w    14142,10000     ;FA#=1.4142
        dc.w    14983,10000     ;SOL=1.4983
        dc.w    15874,10000     ;SOL#=1.5874
        dc.w    16818,10000     ;LA=1.6818
        dc.w    17818,10000     ;LA#=1.7818
        dc.w    18877,10000     ;SI=1.8877

*****************************************************************************

        SECTION Sample,DATA_C   ;venendo letta dal DMA deve essere in CHIP

        ; Armonica di 16 valori creata col'IS del trash'm-one

Armonica:
        DC.B    $19,$46,$69,$7C,$7D,$6A,$47,$1A,$E8,$BB,$97,$84,$83,$95,$B8,$E5

        END

*****************************************************************************
```

Questo sorgente non differisce di molto rispetto al precedente, in quanto include una piccola routine che calcola il periodo di campionamento in base ad una nota data; l'unica differenza e' che ora potrete generare non solo le 7 note di una scala di varie ottave, ma anche le note dei "tasti neri" del pianoforte, ovvero i DIESIS(#)/BEMOLLE(b): in poche parole, non siete limitali

```
ai soli toni, ma avete la possibilita' di suonare anche i SEMITONI.
Innanzitutto, una differenza salta all'occhio: i valori della tabella
"HalfTones" sono assai piu' grandi di quelli della tabella "Notes" dell'esempio
precedente, e questo per garantire una maggiore precisione: infatti, le due
word indicano rispettivamente NUMERATORE e DENOMINATORE della frazione che
indica il rapporto tra una nota ed il DO in una scala, ed il rapporto, appunto,
NON cambia. Prendiamo, a esempio, il SOL: in "Notes" il rapporto e' 3/2 = 1.5,
in "HalfTones" e' pari a 14983/10000 = 1.4983; come vedete il rapporto
e' quasi uguale (lo scarto e' TRASCURABILE).
In "Notes" ho riportato le frazioni "classiche" che si trovano su molti
libri di fisica acustica e che hanno il vantaggio di avere numeratori e
denominatori piccoli e di facile memorizzazione; i valori di "HalfTones",
invece, oltre a riportare i rapporti di tutte le note della scala di semitono
in semitono, hanno una precisione di 4 cifre decimali oltre la "virgola" (che
e' simulata moltiplicando per numeri molto grandi e poi dividento per
10^numero di cifre decimali), ovvero fino ai decimillesimi.
La subroutine "HalfTone2Per" funzione grossomodo come quella "Note2Per";
l'unica differenza sta' nell'esprimere il parametro in ingresso: questa
volta e' necessario indicare il semitono desiderato a partire dal DO1.
Per cui, se vogliamo suonare un FA1 dovremo impostare d0.w=5, poiche'
tra il DO1 ed il FA1 ci sono 5 semitoni di differenza.

In musica, 1 tono = 2 semitoni, ed ogni scala ha 6 toni = 12 semitoni;
tra una nota e l'altra c'e' 1 tono, esclusi gli intervalli di frequenza
tra MI e FA, e tra SI e DO dell'ottava dopo che sono pari ad 1 solo semitono.

Visto che ad ogni ottava e' necessario raddoppiare la frequenza, l'incremento
di frequenza delle note - all'interno di una scala ed oltre - NON e' costante,
ma ESPONENZIALE in base 2.
Per cui, il calcolo dei rapporti delle note all'interno di una scala non e'
poi tanto semplice come potrebbe sembrare: sapendo che l'intervallo dei
rapporti in una scala di 12 semitoni e' pari a 1 (da 1 del primo DO a 2 del
DO dell'ottava successiva), ogni semitono dista dall'altro 1/12 nell'asse
delle ascisse di un grafico cartesiano (x,y) che presenta la funzione
esponenziale: Y = 2^X; consideriamo l'intervallo 0<=X<=1 abbiamo nelle
ordinate un ramo di curva 2^0<=Y<=2^1 = 1<=Y<=2; ora, ad ogni 12esimo
da X=0 calcoliamo il relativo valore in Y, per 12 volte: Y = 2^(1/12),
Y = 2^(2/12), Y = 2^(3/12), e cosi' via fino a Y=2^(12/12) = 2, che
corrisponde al rapporto del 12esimo semitono, ovvero del DO dell'ottava
successiva; ognuno dei valori decimali ottenuti corrisponde al valore
da moltiplicare alla frequenza del DO dell'ottava desiderata per ottenere
la frequenza della nota richiesta nell'ottava medesima, ed e' riconducibile
ad una frazione (anzi, DEVE essere ricondotto ad una frazione con numeri
non decimali perche' il 68000 possa calcolare a numeri interi "simulando"
la virgola).
Per esempio, per sapere il rapporto tra la frequenza di un LA# ed in DO (=1/1):
Y = 2^(10/12) = 2^0.8333 (...periodico...) = 1.7818 (arrotondando); tale numero
decimale e' facilmente riconducibile alla frazione 17818/10000 (si tratta
effettivamente di 17818 decimillesimi)
Ora, se, per esempio, desideriamo un LA3#: DO3 = 131 * 2^(3-1) = 131 * 2 * 2 =
= 131 * 4 = 524 Hz; LA3# = (524 * 17818)/10000 = 933 Hz.
```

Lezione14-3b

```
; Lezione14-3b.s        ** ACCORDI MAGGIORI DI ARMONICHE A 4 VOCI **


        section arm4,code

Start:
```

```
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable
        bset    #1,$bfe001              ; Spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS

        move.l  #armonica,$a0(a6)       ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.l  #armonica,$b0(a6)       ; AUD1LCH.w+AUD1LCL.w=AUD1LC.l
        move.l  #armonica,$c0(a6)       ; AUD2LCH.w+AUD2LCL.w=AUD2LC.l
        move.l  #armonica,$d0(a6)       ; AUD3LCH.w+AUD2LCL.w=AUD3LC.l
        move.w  #16/2,$a4(a6)           ; 16 bytes/2=8 word di dati (AUD0LEN)
        move.w  #16/2,$b4(a6)           ; 16 bytes/2=8 word di dati (AUD1LEN)
        move.w  #16/2,$c4(a6)           ; 16 bytes/2=8 word di dati (AUD2LEN)
        move.w  #16/2,$d4(a6)           ; 16 bytes/2=8 word di dati (AUD3LEN)

        moveq   #16,d1
        moveq   #12*1+0,d2              ;DO2 (accordo di DO)

        move.l  d2,d0
        bsr.s   halftone2per
        move.w  d0,$a6(a6)              ; AUD0PER
        addq.w  #2*2,d2                 ; + 2 toni = MI
        move.l  d2,d0
        bsr.s   halftone2per
        move.w  d0,$b6(a6)              ; AUD1PER
        addq.w  #2+1,d2                 ; + 1 tono + 1 semitono = SOL
        move.l  d2,d0
        bsr.s   halftone2per
        move.w  d0,$c6(a6)              ; AUD2PER
        addq.w  #2+1,d2                 ; + 1 tono + 1 semitono = LA#
        move.l  d2,d0
        bsr.s   halftone2per
        move.w  d0,$d6(a6)              ; AUD3PER

        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #64,$b8(a6)             ; AUD1VOL al massimo (0 dB)
        move.w  #64,$c8(a6)             ; AUD2VOL al massimo (0 dB)
        move.w  #64,$d8(a6)             ; AUD3VOL al massimo (0 dB)
        move.w  #$800f,$96(a6)          ; Accende AUD0-AUD3 DMA in DMACONW

WLMB:
        btst    #6,$bfe001              ;aspetta il tasto sinistro del mouse
        bne.s   WLMB
        or.w    #$8000,d7               ; accende il bit 15 (SET/CLR)
        move.w  #$000f,$96(a6)          ; spegne i DMA
        move.w  d7,$96(a6)              ; reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts

*****************************************************************************
;                       « HalfTone To Period »
;
; Calcola il periodo da inserire in AUDxPER data il semitono a partire dal DO1
;
; d0.w   = semitono (a partire dal DO1=0)
; d1.w   = lunghezza armonica (in byte)
*****************************************************************************

Clock   equ     3546895
DO1     equ     131                     ; Frequenza [Hz] del DO 1a ottava
```

```
HalfTone2Per:
        move.l  d2,-(SP)
        divu.w  #12,d0
        move.w  #do1,d2
        lsl.w   d0,d2
        swap    d0
        add.w   d0,d0
        add.w   d0,d0
        mulu.w  halftones(pc,d0.w),d2
        divu.w  halftones+2(pc,d0.w),d2
        move.l  #clock,d0
        mulu.w  d2,d1
        divu.w  d1,d0           ; DIVISION BY ZERO!!!
        move.l  (SP)+,d2
        rts                     ; [d0.w=periodo di campionamento]

HalfTones:
        dc.w    10000,10000     ;DO=1.0
        dc.w    10595,10000     ;DO#=1.0595
        dc.w    11225,10000     ;RE=1.1225
        dc.w    11892,10000     ;RE#=1.1892
        dc.w    12599,10000     ;MI=1.2599
        dc.w    13348,10000     ;FA=1.3348
        dc.w    14142,10000     ;FA#=1.4142
        dc.w    14983,10000     ;SOL=1.4983
        dc.w    15874,10000     ;SOL#=1.5874
        dc.w    16818,10000     ;LA=1.6818
        dc.w    17818,10000     ;LA#=1.7818
        dc.w    18877,10000     ;SI=1.8877

*****************************************************************************

        SECTION Sample,DATA_C   ;venendo letta dal DMA deve essere in CHIP

        ; Armonica di 16 valori creata col'IS del trash'm-one

Armonica:
        DC.B    $19,$46,$69,$7C,$7D,$6A,$47,$1A,$E8,$BB,$97,$84,$83,$95,$B8,$E5

        END

*****************************************************************************
```

Questo sorgente non differisce per nulla rispetto a quello precedente.
L'unica novita' introdotta e' stata l'uso di tutte le voci hardware del chip sonoro dell'Amiga...niente di complicato, in realta': per suonare lo stesso sample a frequenza diversa e' sufficiente impostare tutti i registri AUDxLC, AUDxLEN ed AUDxVOL con il medesimo valore per tutti i canali, e variare solo i periodi per gli AUDxPER.

In musica, per creare un ACCORDO MAGGIORE a 3 o piu' note (noi lo abbiamo fatto a 4, tanto per non lasciar oziare l'ultima voce...), bisogna suonare CONTEMPORANEAMENTE 3 tutte le note giuste che formano l'accordo.
Ecco lo schema generale:

```
                            ************ ACCORDI MAGGIORI ***********
                            +------+--------------------------------+
                            | NOTA |       TONALITA'                |
                            +------+--------------------------------+
                            |  1a  |  nota di base dell'accordo     |
                            |  2a  |  + 2 toni = 4 semitoni         |
                            |  3a  |  + 1 tono e mezzo = 3 semitoni |
```

```
                              |  4a  |  + 1 tono e mezzo = 3 semitoni |
                              +------+--------------------------------+
```

Per esempio, per l'accordo di MI a 3 voci: MI + SOL# + SI; per l'accordo di LA a 4 voci: LA + DO# + MI + SOL.

28.4 Lezione14-4a

```
; Lezione14-4a.s        ** SUONA FORME D'ONDA COMPLESSE **


        section samplemono,code

Start:
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable
        bset    #1,$bfe001              ; Spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS

        move.l  #sample,$a0(a6)         ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.l  #sample,$b0(a6)         ; AUD1LCH.w+AUD1LCL.w=AUD1LC.l
        move.w  #(sample_end-sample)/2,$a4(a6)  ; lunghezza in word (AUD0LEN)
        move.w  #(sample_end-sample)/2,$b4(a6)  ; lunghezza in word (AUD1LEN)

Clock   equ     3546895

        move.w  #clock/21056,$a6(a6)    ; AUD0PER a 168
        move.w  #clock/21056,$b6(a6)    ; AUD1PER a 168

        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #64,$b8(a6)             ; AUD1VOL al massimo (0 dB)
        move.w  #$8003,$96(a6)          ; Accende AUD0-AUD1 DMA in DMACONW


WLMB:
        btst    #6,$bfe001              ;aspetta il tasto sinistro del mouse
        bne.s   WLMB

        or.w    #$8000,d7               ; accende il bit 15 (SET/CLR)
        move.w  #$0003,$96(a6)          ; spegne i DMA
        move.w  d7,$96(a6)              ; reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts

*****************************************************************************

        SECTION Sample,DATA_C

; Nota: il sample e' tratto da "NASP" by Pyratronik/IBB

Sample: incbin  "assembler2:sorgenti8/carrasco.21056"
Sample_end:

        END

*****************************************************************************
```

Per quanto riguarda questo esempio, le cose da spiegare non sono poi molte:

```
non ci sono novita', anzi, e' molto simile all'esempio 1, e siamo abituati
a listati ben piu' impegnativi.
Preciso solamente una cosa: la frequenza di campionamento del sample e' di
21056 Hz, pari alla frequenza originale di registrazione: e' necessario
porre una VELOCITA' DI CAMPIONAMENTO uguale a quella di digitalizzazione
se si vuole sentire il suono alla velocita' corretta...provate a cambiare
il periodo di campionamento in AUDxPER...

*** Voglio sottolineare che 21056 NON esprime il numero di volte in cui
viene letto l'intero sample, ma la frequenza di lettura di byte per byte:
vengono letto 21056 byte al secondo in un sample di lunghezza arbitraria;
all'hardware bisogna comunicare il periodo di campionamento relativo
alla velocita' di lettura.
Come abbiamo fatto per l'armonica: prima abbiamo stabilito quante volte doveva
venir letta l'INTERA onda, poi abbiamo calcolato il periodo di campionamento
moltiplicando la frequenza della nota per la lunghezza del sample in byte, per
ottenere la velocita' di lettura ***.
```

Lezione14-4b

```
; Lezione14-4b.s        ** SUONA PIU' FORME D'ONDA COMPLESSE **

        section samplestereo,code

Start:
        move.l  4.w,a6
        jsr     -$78(A6)                ; _LVODisable
        bset    #1,$bfe001              ; Spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ; dmaconr - Salva DMA dell'OS

        move.l  #sample1,$a0(a6)        ; AUD0LCH.w+AUD0LCL.w=AUD0LC.l
        move.l  #sample2,$b0(a6)        ; AUD1LCH.w+AUD1LCL.w=AUD1LC.l
        move.w  #(sample1_end-sample1)/2,$a4(a6) ; lunghezza in word (AUD0LEN)
        move.w  #(sample2_end-sample2)/2,$b4(a6) ; lunghezza in word (AUD1LEN)

Clock   equ     3546895

        move.w  #clock/21056,$a6(a6)    ; AUD0PER a 168
        move.w  #clock/21056,$b6(a6)    ; AUD1PER a 168

        move.w  #64,$a8(a6)             ; AUD0VOL al massimo (0 dB)
        move.w  #64,$b8(a6)             ; AUD1VOL al massimo (0 dB)
        move.w  #$8003,$96(a6)          ; Accende AUD0-AUD1 DMA in DMACONW
WLMB:
        btst    #6,$bfe001              ; Aspetta il tasto sinistro del mouse
        bne.s   WLMB

        or.w    #$8000,d7               ; accende il bit 15 (SET/CLR)
        move.w  #$0003,$96(a6)          ; spegne i DMA
        move.w  d7,$96(a6)              ; reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     -$7e(a6)                ; _LVOEnable
        rts

*****************************************************************************

        SECTION Sample,DATA_C

; Nota: i sample sono tratti da "NASP" by Pyratronik/IBB
```

```
Sample1:
        incbin  "assembler2:sorgenti8/carrasco.21056"
Sample1_end:

Sample2:
        incbin  "assembler2:sorgenti8/lee3.21056"
Sample2_end:

        END

*****************************************************************************

Abbiamo semplicemente suonato due sample diversi in stereo, due sample
che avevano la stessa frequenza di lettura ideale (ma potrebbero averla avuta
anche differente: non sarebbe cambiato nulla !) e la stessa lunghezza (cosa
molto importante, poiche' letti alla stessa frequenza hanno la medesima
durata e loopano sincronizzati).
```

28.5 Lezione14-5a

```
; Lezione14-5a.s        ** SUONA SAMPLE MOLTO LUNGHI **


        SECTION PlayLongSamples,CODE

Start:
        bset    #1,$bfe001              ; spegne il filtro passa-basso

                                        ; >>>> PARAMETRI <<<<
        lea     sample,a0               ; indirizzo sample
        move.l  #sample_end-sample,d0   ; lunghezza sample in byte
        move.w  #17897,d1               ; frequenza di lettura
        moveq   #64,d2                  ; volume
        bsr.s   playlongsample_init     ; INIT routine (comincia)....
                                        ; ....CPU libera....
WLMB:
        btst    #6,$bfe001              ; testa LMB+RMB...provate, dunque,
        bne.s   wlmb                    ; a girare per il Wb e noterete
        btst    #10,$dff016             ; come non avvenga NESSUN rallentamento
        bne.s   wlmb                    ; ....magie del DMA !

        bsr.w   playlongsample_restore  ; RESTORE routine (spegne tutto)
        rts


*************************************
*****  Play Long Sample Routines  *****
*************************************
;
; a0    = sample adr
; d0.l  = lunghezza.b sample, d1.w=frequenza, d2.w=volume
;
; L'AutoVector Lv4 IRQ deve essere disponibile


_LVOSupervisor  equ     -30
Clock           equ     3546895
AFB_68010       equ     0
AttnFlags       equ     296
```

```
PlayLongSample_init:
        movem.l d2/a0/a6,-(sp)
        movem.l d0/a0,plsregs           ; registri fissi di riferimento
        movem.l d0/a0,plsregs+4*2       ; registri di lavoro
        sub.l   a0,a0                   ; FAST CLEAR An
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)      ; 68010+ ?
        beq.s   .no010
        lea     getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010:
        lea     $dff000,a6
        move.w  #$0780,$9c(a6)          ; azzera eventuali richieste di IRQ
        move.w  $1c(a6),oldint          ; salva INTENA dell'OS
        move.w  #$0780,$9a(a6)          ; maschera INT AUD0-AUD3
        move.l  $70(a0),oldlv4          ; salva l'autovettore del livello 4
        move.l  #lv4irq,$70(a0)         ; imposta il nuovo autovettore
        move.w  d2,$a8(a6)              ; imposta AUD0VOL
        move.w  d2,$b8(a6)              ; imposta AUD1VOL
        move.w  d2,$c8(a6)              ; imposta AUD2VOL
        move.w  d2,$d8(a6)              ; imposta AUD3VOL
        move.l  #clock,d2
        divu.w  d1,d2                   ; d2.w=clock/freq = periodo di camp.
        move.w  d2,$a6(a6)              ; imposta AUD0PER
        move.w  d2,$b6(a6)              ; imposta AUD1PER
        move.w  d2,$c6(a6)              ; imposta AUD2PER
        move.w  d2,$d6(a6)              ; imposta AUD3PER
        move.w  $2(a6),olddma           ; salva DMACON dell'OS
        move.w  #$8400,$9a(a6)          ; accende AUD3 IRQ - basta lui...
        move.w  #$8400,$9c(a6)          ; forza l'IRQ per comiciare...
        movem.l (sp)+,d2/a0/a6
        rts

;---------------------------------------
GetVBR:
        dc.l    $4e7a8801       ;movec  vbr,a0  ;base dei vettori di eccezione
        rte
;---------------------------------------

PlayLongSample_restore:
        movem.l d0/a0/a6,-(sp)
        sub.l   a0,a0
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)
        beq.s   .no010
        lea     getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010:
        lea     $dff000,a6
        move.w  #$0780,$9c(a6)          ; azzera le richieste di tutti i canali
        move.w  #$0400,$9a(a6)          ; maschera l'INT AUD3
        move.l  oldlv4(pc),$70(a0)      ; reimposta l'autovettore 4 dell'OS
        move.w  #$000f,$96(a6)          ; spegne tutti i DMA audio
        move.w  oldint(pc),d0
        or.w    #$8000,d0               ; imposta SET/CLR che e' a 0 in INTENAR
        move.w  d0,$9a(a6)              ; reimposta l'INTENA dell'OS
        move.w  olddma(pc),d0
        or.w    #$8000,d0               ; imposta SET/CLR che e' a 0 in DMACONR
        move.w  d0,$96(a6)              ; reimposta il DMACON dell'OS
        movem.l (sp)+,d0/a0/a6
        rts
```

```
;---------------------------------------

PlayLongSample_IRQ:
        movem.l d0-d1/a0-a1/a6,-(sp)
        lea     $dff000,a6
        movem.l plsregs+4*2(pc),d0/a0   ; grabba i registri di lavoro
        move.l  a0,$a0(a6)              ; imposta AUD0LC
        move.l  a0,$b0(a6)              ; imposta AUD1LC
        move.l  a0,$c0(a6)              ; imposta AUD2LC
        move.l  a0,$d0(a6)              ; imposta AUD3LC
        move.l  d0,d1                   ; d1.l=lunghezza mancante
        and.l   #~(128*1024-1),d1       ; mancano ancora piu' di 128 kB
        bne.s   .long                   ; se SI: vai a .long
        move.l  d0,d1                   ; se NO: usa lungh. mancante (< 128 kB)
.Long:  lsr.l   #1,d1                   ; trasforma la lungh. da suonare in .W
        move.w  d1,$a4(a6)              ; imposta AUD0LEN
        move.w  d1,$b4(a6)              ; imposta AUD1LEN
        move.w  d1,$c4(a6)              ; imposta AUD2LEN
        move.w  d1,$d4(a6)              ; imposta AUD3LEN
        add.l   #128*1024,a0            ; punta a0 al prossimo blocco
        sub.l   #128*1024,d0            ; lunghezza MENO 128 kB
        bhi.s   .noloop                 ; d0 => 1 ? (manca ALMENO 1 byte)
        movem.l plsregs(pc),d0/a0       ; se NO: reimposta registri originali
.NoLoop:movem.l d0/a0,plsregs+4*2       ; salva comunque d0 e a0 nelle copie
        move.w  #$820f,$96(a6)          ; accende tutti i DMA audio, e viene
                                        ; generato subito l'IRQ, nel caso
                                        ; della prima accensione dell'audio
        movem.l (sp)+,d0-d1/a0-a1/a6
        rts

;---------------------------------------

OldINT: dc.w    0
OldDMA: dc.w    0
OldLv4: dc.l    0
PLSRegs: dc.l   0,0     ; lunghezza,puntatore - fissi
        dc.l    0,0     ; lunghezza,puntatore - variabili


**************************************
*****  Level 4 Interrupt Handler  *****
**************************************

        cnop    0,8
Lv4IRQ:
        btst    #10-8,$dff01e           ;IRQ AUD3 ?
        beq.s   .exit                   ;se NO: esci
        move.w  #$0400,$dff09c          ;spegni subito la richiesta, poiche'
                                        ;nella routine vengono accesi i DMA
                                        ;ed il nuovo IRQ viene subito generato:
                                        ;spegnendo la richiesta dopo la routine
                                        ;si corre il rischio di annullare la
                                        ;richiesta di IRQ al primo ciclo
                                        ;dell'interrupt (appena avviata la
                                        ;routine).
        bsr.w   playlongsample_irq
.Exit:
        rte
```

```
        SECTION Sample,DATA_C

        ; MammaGamma by Alan Parsons Project (©1981)
Sample:
        incbin  "assembler2:sorgenti8/Mammagamma.17897"
Sample_end:

        END
```

Ora le cosa ricominciano a complicarsi... Abbiamo iniziato ad usare interrupt
e routine non piu' - diciamo - banali.
Come e' stato gia' scritto nella LEZIONE, i canali audio sono associati a 4
diversi interrupt assegnati al livello 4 del 680x0; tali interrupt vengono
generati dall'harware ogni qual volta un canale viene forzato a leggere i dati
in memoria a partire dall'indirizzo contenuto nel suo AUDLC: avvegono, dunque,
appena acceso il DMA ed ogni volta all'inizio di un nuovo loop del sample.
* Appena un canale comincia a suonare dall'inizio un sample, oltre a venir
"sparato" l'IRQ, il suo AUDLC rimane immutato e, pertanto, ALTERABILE:
in questo modo funziona la routine "PlayLongSample": ogni volta che il DMA
comincia a leggere un pezzo (da 128 kB o meno, a seconda che la parte di
sample mancante da suonare sia piu' lunga della massima AUDLEN loopabile) viene
generato l'interrupt, ed i registri di locazione AUDxLC (tutti e 4, in questo
caso, poiche' vengono utilizzati tutti per suonare i medesimi dati) vengono
ricalcolati e fatti avanzare o retrocedere in base al "pezzetto" di sample
a cui PRECEDENTEMENTE putavano e che ORA il DMA sta' leggendo *.
*** In sostanza, con questa tecnica, e' possibile far suonare all'Amiga tanti
pezzi di 128 kB - o meno, per quanto riguarda l'ultimo pezzo - di dati audio
adiacenti in memoria, senza far sentire lo "stacco" tra uno e l'altro ***.

N.B.:   una volta avviata la routine, essa continua a far loopare il sample
        all'infinito solo dal codice in interrupt, per cui ** e' COMPLETAMENTE
        INDIPENDENTE: in altre parole, dopo l'"_init", tornate in main
        ed avete normale controllo di tutto l'hardware (escluso quello sonoro,
        ovviamente) e della CPU (escluso l'interrupt di livello 4, che e'
        quello utilizzato dalla "PlayLongSample") **.
        Quando volete spegnere il sample, chiamate la "_restore" e tutto
        tornera' come prima di aver chiamato la "_init" (eventuali routine
        su interrupt audio comprese !).

P.S.:   un'ultima precisazione: qui e' stato utilizzato solo un IRQ per
        tutte le voci, visto che tutte suonavano contemporaneamente la
        medesima cosa, e precisamente quello della voce 3, ovvero il piu'
        alto di priorita' hardware.
        In teoria non dovrebbe cambiare nulla nell'usare quello di quelche
        altra voce, * a patto che si mascherino gli altri - o che comunque
        vengano ignorati dall'handler -, altrimenti al termine di ogni blocco
        letto verrebbero generati 4 interrupt.

Lezione15-5-b

```
; Corso Asm - LEZIONE xx:  ** SUONA SAMPLE MOLTO LUNGHI SOTTO OS **


        SECTION PlayLongSamples_OS,CODE

Start:
        bset    #1,$bfe001

        lea     sample,a0
```

```
        move.l  #sample_end-sample,d0
        move.w  #17897,d1
        moveq   #64,d2
        bsr.s   playlongsample_init

WLMB:   btst    #6,$bfe001
        bne.s   wlmb
        btst    #10,$dff016
        bne.s   wlmb

        bsr.w   playlongsample_restore
        rts


**************************************
*****  Play Long Sample Routines  *****
**************************************

PlayLongSample_init:
                ;[a0=sample adr]
                ;[d0.l=lunghezza.b sample, d1.w=frequenza, d2.w=volume]

Clock           equ     3546895
NT_Interrupt    equ     2
LN_Type         equ     8
LN_Pri          equ     9
LN_Name         equ     10
IS_Data         equ     14
IS_Code         equ     18
IS_SIZE         equ     22
_LVOSetIntVector        equ     -162

        movem.l d0/d2/a1/a6,-(sp)
        movem.l d0/a0,plsregs
        movem.l d0/a0,plsregs+4*2
        movem.l d1-d2,-(sp)
        move.l  4.w,a6                          ;base di exec in a6
        lea     aud1int_node(pc),a1             ;struttura/nodo dell'interrupt
        move.b  #nt_interrupt,ln_type(a1)       ;tipo di nodo: interrrupt
        move.l  #aud1int_name,ln_name(a1)       ;nome del nodo public
        move.l  #aud1int_data,is_data(a1)       ;punta ai dati (a1-scratch)
        move.l  #aud1int_code,is_code(a1)       ;punta al codice (a5-scratch)
        moveq   #8,d0                           ;bit di INTENA/INTREQ (AUD1)
        jsr     _LVOSetIntVector(a6)
        move.l  d0,oldaud1int_node              ;d0.l=nodo precedente
        movem.l (sp)+,d1-d2
        lea     $dff000,a6
        move.w  d2,$a8(a6)
        move.w  d2,$b8(a6)
        move.w  d2,$c8(a6)
        move.w  d2,$d8(a6)
        move.l  #clock,d2
        divu.w  d1,d2
        move.w  d2,$a6(a6)
        move.w  d2,$b6(a6)
        move.w  d2,$c6(a6)
        move.w  d2,$d6(a6)
        move.w  $2(a6),olddma
        move.w  $1c(a6),oldint
        move.w  #$8100,$9a(a6)
        move.w  #$8100,$9c(a6)
        movem.l (sp)+,d0/d2/a1/a6
```

```
        rts
;-------------------------------------
PlayLongSample_restore:
        movem.l d0/a1/a6,-(sp)
        move.l  4.w,a6
        move.l  oldaud1int_node(pc),a1          ;reimposta nodo precedente
        moveq   #8,d0                           ;bit di INTENA/INTREQ (AUD1)
        jsr     _LVOSetIntVector(a6)
        lea     $dff000,a6
        move.w  #$0780,$9c(a6)                  ;spegne tutte le richieste IRQ
        move.w  #$0100,$9a(a6)
        move.w  oldint(pc),d0
        or.w    #$8000,d0
        move.w  d0,$9a(a6)
        move.w  #$000f,$96(a6)
        move.w  olddma(pc),d0
        or.w    #$8000,d0
        move.w  d0,$96(a6)
        movem.l (sp)+,d0/a1/a6
        rts
;-------------------------------------
PlayLongSample_IRQ:                     ;<<< questa routine e' identica
        movem.l d0-d1/a0-a1/a6,-(sp)
        lea     $dff000,a6
        movem.l plsregs+4*2(pc),d0/a0
        move.l  a0,$a0(a6)
        move.l  a0,$b0(a6)
        move.l  a0,$c0(a6)
        move.l  a0,$d0(a6)
        move.l  d0,d1
        and.l   #~(128*1024-1),d1
        bne.s   .long
        move.l  d0,d1
.Long:  lsr.l   #1,d1
        move.w  d1,$a4(a6)
        move.w  d1,$b4(a6)
        move.w  d1,$c4(a6)
        move.w  d1,$d4(a6)
        add.l   #128*1024,a0
        sub.l   #128*1024,d0
        bhi.s   .noloop
        movem.l plsregs(pc),d0/a0
.NoLoop:movem.l d0/a0,plsregs+4*2
        move.w  #$820f,$96(a6)
        movem.l (sp)+,d0-d1/a0-a1/a6
        rts
;-------------------------------------
OldDMA: dc.w    0
OldInt: dc.w    0
OldAud1Int_Node:dc.l    0
Aud1Int_Node:
        blk.b   is_size         ;lunghezza InterruptStructure
        even
Aud1Int_Name:
        dc.b    "PlayLongSampleIRQ",0
        even
Aud1Int_Data:
PLSRegs:dc.l    0,0     ;lunghezza,puntatore - fissi
        dc.l    0,0     ;lunghezza,puntatore - variabili

        cnop   0,8
Aud1Int_Code:
```

```
        move.w  #$0100,$dff09c
        bsr.w   playlongsample_irq
        rts



        SECTION Sample,DATA_C

        ; MammaGamma by Alan Parsons Project (©1981)
Sample:
        incbin  "assembler2:sorgenti8/Mammagamma.17897"
Sample_end:

        END
```

```
Questa volta non e' cambiato quasi nulla rispetto al sorgente precedente:
abbiamo solo allocato l'handler di interrupt con l'exec library, in modo
da rendere tutto un po' piu' "friendly" nei confronti del sistema operativo.

N.B.:   e' stato utilizzato l'interrupt del canale 1, poiche', per le pseudo
        priorita' software dell'exec, e' il primo a essere rilevato nello
        handler interno in ROM di livello 4.

P.S.:   una precisazione per quanto riguarda la differenza tra Server Chain
        ed Handler di interrupt per l'exec: certi interrupt (VERTB, COPER,
        PORTS, EXTER e NMI) sono piu' utili di altri e vengono usati spesso
        sia dall'OS che dai task utente; l'exec deve pertanto dare la
        possibilita' a tutti di avere delle proprie routine in interrupt, e
        forma quindi delle "catene" di routine aventi diversa e specificabile
        priorita' di esecuzione gestite da un unico handler.
        Tutti gli altri interrupt del Paula (TBE, DSKBLK, SOFT, BLIT, AUD0-3,
        RBF e DSKSYNC) non sono visti come server chain ma come handler: ognuno
        puo' impossessarsi del dato interrupt completamente, senza linkarsi
        o dividerselo con nessun altro task.
        Nel nostro caso, abbiamo alloccato l'interrupt del canale 1, quello a
        priorita' software maggiore, per l'exec (...e non chiedetemi perche'),
        con _LVOSetIntVector perche' richiede un handler, non un server;
        inoltre, nel caso degli handler, la priorita' del nodo della struturra
        dell'interrupt non necessita di essere impostata poiche' non
        vi sono altri server nella chain, si e' soli.

P.P.S.: tutte le note del sorgente precedente - a parte quelle variate -
        valgono anche per questo.

        N.B.:   gli EQU provengono dagli include "exec/interrupt.i" ed
                "LVO1.3/exec_lib.i".
```

Lezione14-5c

```
; Corso Asm - LEZIONE xx:  ** SUONA SAMPLE MOLTO LUNGHI **

; DA DEBUGGARE

WLMB    macro
\@:     btst    #6,$bfe001
        bne.s   \@
        endm

WRMB    macro
```

```
\@:     btst    #10,$dff016
        bne.s   \@
        endm


        SECTION PlayLongSamples,CODE

Start:
        bset    #1,$bfe001              ;spegne il filtro passa-basso
                                        ;>>>> PARAMETRI <<<<
        lea     sample,a0               ;indirizzo sample
        move.l  #sample_end-sample,d0   ;lunghezza sample in byte
        move.w  #17897,d1               ;frequenza di lettura
        moveq   #64,d2                  ;volume

        moveq   #0,d3                   ;suona voce 0
        bsr.s   playlongsample_init
        WLMB
        moveq   #1,d3                   ;suona voce 1
        bsr.s   playlongsample_init
        WRMB
        moveq   #2,d3                   ;suona voce 2
        bsr.s   playlongsample_init
        WLMB
        moveq   #3,d3                   ;suona voce 3
        bsr.s   playlongsample_init
        WRMB

        moveq   #0,d3                   ;spegni voce 0
        bsr.w   playlongsample_restore
        WLMB
        moveq   #1,d3                   ;spegni voce 1
        bsr.w   playlongsample_restore
        WRMB
        moveq   #2,d3                   ;spegni voce 2
        bsr.w   playlongsample_restore
        WLMB
        moveq   #3,d3                   ;spegni voce 3
        bsr.w   playlongsample_restore
        rts


*************************************
*****  Play Long Sample Routines  *****
*************************************

PlayLongSample_init:
                ;[a0=sample adr]
                ;[d0.l=lunghezza.b sample, d1.w=frequenza, d2.w=volume]
                ;[d3.w=voce (0..3)]
                ;* L'AutoVettore Lv4 IRQ deve essere disponibile *

Clock           equ     3546895
_LVOSupervisor  equ     -30
AFB_68010       equ     0
AttnFlags       equ     296

        movem.l d0-d7/a0-a1/a5-a6,-(sp)
        and.w   #3,d3                   ;al massimo 3 canali
        lea     $dff000,a6
        moveq   #1,d4
        lsl.w   d3,d4
```

```
        move.w  d4,d6
        and.w   $2(a6),d4               ;maschera DMA della voce
        move.w  #1<<7,d5
        lsl.w   d3,d5
        move.w  d5,d7
        and.w   $1c(a6),d5              ;maschera INT della voce
        add.w   d3,d3                   ;d3=d3*2: esprime offset di word
        lea     olddmas(pc),a1
        move.w  d4,(a1,d3.w)            ;salva stato vecchio del DMA della voce
        lea     oldints(pc),a1
        move.w  d5,(a1,d3.w)            ;salva stato vecchio del INT della voce
        move.w  d7,$9c(a6)              ;azzera eventuali IRQ
        move.w  d6,$96(a6)              ;spegni DMA della voce
        move.w  d7,$9a(a6)              ;spegni INT della voce
        sub.l   a1,a1                           ;FAST CLEAR An
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)      ;68010+ ?
        beq.s   .no010
        lea     .getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010: cmp.l   #lv4irq,$70(a1)
        beq.s   .nochg
        move.l  $70(a1),oldlv4          ;salva l'autovettore del livello 4
        move.l  #lv4irq,$70(a1)         ;imposta il nuovo autovettore
.NoChg: lsl.w   #4-1,d3                 ;d3=d3*8: ora esprime offset di 16 byte
        lea     $dff0a0,a6
        move.w  d2,$8(a6,d3.w)          ;imposta AUDxVOL
        move.l  #clock,d2
        divu.w  d1,d2                   ;d2.w=clock/freq = periodo di camp.
        move.w  d2,$6(a6,d3.w)          ;imposta AUDxPER
        lea     $dff000,a6
        or.w    #$8000,d7
        move.w  d7,$9a(a6)              ;accende INT della voce
        lea     plsregs(pc,d3.w),a1
        movem.l d0/a0,(a1)              ;registri fissi
        movem.l d0/a0,4*2(a1)           ;registri di lavoro
        move.w  d7,$9c(a6)              ;forza IRQ della voce...
        movem.l (sp)+,d0-d7/a0-a1/a5-a6
        rts
.GetVBR:
        dc.l    $4e7a9801       ;movec  vbr,a1  ;base dei vettori di eccezione
        rte
;---------------------------------------
PLSRegs:        ;DEVONO STARE TRA _INIT E _IRQ PER IL MODO DI INDIRIZZAMENTO
                ;USATO: XX(pc,Rn) CHE CONSENTE SOLO 8 BIT CON SEGNO AD "XX"
PLSAud0Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
PLSAud1Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
PLSAud2Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
PLSAud3Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
;---------------------------------------
PlayLongSample_IRQ:
                ;[a1=PLSAudxRegs]
                ;[d3.w=voce]
        movem.l d0-d3/a0-a1/a6,-(sp)
        and.w   #3,d3                   ;al massimo 3 voci
        move.w  d3,d2
        lsl.w   #4,d3                   ;d3=d3*16: esprime offset di 16 byte
        lea     plsregs(pc,d3.w),a1
```

```
        movem.l 4*2(a1),d0/a0           ;grabba i registri di lavoro
        lea     $dff0a0,a6
        move.l  a0,$0(a6,d3.w)          ;imposta AUDxLC
        move.l  d0,d1                   ;d1.l=lunghezza mancante
        and.l   #~(128*1024-1),d1       ;mancano ancora piu' di 128 kB
        bne.s   .long                   ;se SI: vai a .long
        move.l  d0,d1                   ;se NO: usa lungh. mancante (< 128 kB)
.Long:  lsr.l   #1,d1                   ;trasforma la lungh. da suonare in WORD
        move.w  d1,$4(a6,d3.w)          ;imposta AUDxLEN
        add.l   #128*1024,a0            ;punta a0 al prossimo blocco
        sub.l   #128*1024,d0            ;lunghezza MENO 128 kB
        bhi.s   .noloop                 ;d0 => 1 ? (manca ancora ALMENO 1 byte)
        movem.l (a1),d0/a0              ;se NO: reimposta registri originali
.NoLoop:movem.l d0/a0,4*2(a1)           ;salva comunque d0 e a0 nelle copie
        move.w  #%1<<7,d0
        lsl.w   d2,d0
        move.w  d0,$dff09c              ;azzera IRQ della voce per non subire
                                        ;un nuovo interrupt appena uscito
        moveq   #%1,d0
        lsl.w   d2,d0
        or.w    #$8200,d0               ;accende DMA della voce
        move.w  d0,$dff096
        movem.l (sp)+,d0-d3/a0-a1/a6
        rts
;-------------------------------------
PlayLongSample_restore:
                ;[d3.w=voce (0..3)]
        movem.l d0-d1/d3/a0/a6,-(sp)
        and.w   #3,d3                   ;al massimo 3 voci
        lea     $dff000,a6
        moveq   #1,d0
        lsl.w   d3,d0
        move.w  #1<<7,d1
        lsl.w   d3,d1
        move.w  d1,$9c(a6)              ;azzera eventuali IRQ della voce
        move.w  d1,$9a(a6)              ;spegne INT della voce
        move.w  d0,$96(a6)              ;spegne DMA della voce
        move.w  $1c(a6),d0
        and.w   #$0780,d0               ;spente tutte le voci = ultima voce ?
        bne.s   .NoOFF
        sub.l   a0,a0                   ;se SI:...
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)
        beq.s   .no010
        lea     .getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010: move.l  oldlv4(pc),$70(a0)      ;...reimposta il vecchio autovettore
.NoOFF: lea     $dff000,a6
        add.w   d3,d3                   ;d3=d3*2: esprime offset di word
        move.w  oldints(pc,d3.w),d0
        or.w    #$8000,d0
        move.w  d0,$9a(a6)              ;accende vecchi INT
        move.w  olddmas(pc,d3.w),d0
        or.w    #$8000,d0
        move.w  d0,$96(a6)              ;accende vecchi DMA
        movem.l (sp)+,d0-d1/d3/a0/a6
        rts
.GetVBR:
        dc.l    $4e7a8801       ;movec  vbr,a0  ;base dei vettori di eccezione
        rte
;-------------------------------------
OldINTs:dc.w    0,0,0,0
```

```
OldDMAs:dc.w    0,0,0,0
OldLv4: dc.l    0


*************************************
*****  Level 4 Interrupt Handler  *****
*************************************

        cnop    0,8
Lv4IRQ:
        move.w  d3,-(sp)
        link    a0,#0
.IRQLp: pea     .irqlp(pc)              ;pusha il ritorno per l'RTS nello stack
        moveq   #3,d3
        btst    #10-8,$dff01e           ;aud3 IRQ ?
        bne.w   playlongsample_irq      ;se SI: bracha (seza ritorno) alla _IRQ
        moveq   #2,d3
        btst    #9-8,$dff01e            ;aud2 IRQ ?
        bne.w   playlongsample_irq
        moveq   #1,d3
        btst    #8-8,$dff01e            ;aud1 IRQ ?
        bne.w   playlongsample_irq
        moveq   #0,d3                   ;aud0 IRQ ?
        btst    #7,$dff01f
        bne.w   playlongsample_irq
        unlk    a0
        move.w  (sp)+,d3
        rte


        SECTION Sample,DATA_C

        ; MammaGamma by Alan Parsons Project (©1981)
Sample:
        incbin  "assembler2:sorgenti8/Mammagamma.17897"
Sample_end:

        END
```

Lezione14-5d

```
; Corso Asm - LEZIONE xx:  ** EFFETTO FAKE-SURROND **
                        ; N.B.: funziona bene sui computer lenti per il
                        ;       ritardo tra le due voci per cassa.

; debug, stesso di prima

WLMB    macro
\@      btst    #6,$bfe001
        bne.s   \@
        endm

WRMB    macro
\@      btst    #10,$dff016
        bne.s   \@
        endm

        SECTION PlayLongSamples,CODE

Start:
```

```
        bset    #1,$bfe001              ;spegne il filtro passa-basso
                                        ;>>>> PARAMETRI <<<<
        lea     sample,a0               ;indirizzo sample
        move.l  #sample_end-sample,d0   ;lunghezza sample in byte
        move.w  #17897,d1               ;frequenza di lettura
        moveq   #64,d2                  ;volume

        moveq   #0,d3                   ;suona voce 0
        bsr.w   playlongsample_init
        moveq   #3,d3                   ;suona voce 3
        bsr.w   playlongsample_init
        WLMB
        moveq   #1,d3                   ;suona voce 1
        bsr.s   playlongsample_init
        moveq   #2,d3                   ;suona voce 2
        bsr.s   playlongsample_init
        WRMB

        moveq   #0,d3                   ;spegni voce 0
        bsr.w   playlongsample_restore
        moveq   #1,d3                   ;spegni voce 1
        bsr.w   playlongsample_restore
        moveq   #2,d3                   ;spegni voce 2
        bsr.w   playlongsample_restore
        moveq   #3,d3                   ;spegni voce 3
        bsr.w   playlongsample_restore
        rts



*************************************
*****  Play Long Sample Routines  *****
*************************************

PlayLongSample_init:
                ;[a0=sample adr]
                ;[d0.l=lunghezza.b sample, d1.w=frequenza, d2.w=volume]
                ;[d3.w=voce (0..3)]
                ;* L'AutoVettore Lv4 IRQ deve essere disponibile *

_LVOSupervisor  equ     -30
Clock           equ     3546895
AFB_68010       equ     0
AttnFlags       equ     296

        movem.l d0-d7/a0-a1/a6,-(sp)
        and.w   #3,d3                   ;al massimo 3 canali
        lea     $dff000,a6
        moveq   #1,d4
        lsl.w   d3,d4
        move.w  d4,d6
        and.w   $2(a6),d4               ;maschera DMA della voce
        move.w  #1<<7,d5
        lsl.w   d3,d5
        move.w  d5,d7
        and.w   $1c(a6),d5              ;maschera INT della voce
        add.w   d3,d3                   ;d3=d3*2: esprime offset di word
        lea     olddmas(pc),a1
        move.w  d4,(a1,d3.w)            ;salva stato vecchio del DMA della voce
        lea     oldints(pc),a1
        move.w  d5,(a1,d3.w)            ;salva stato vecchio del INT della voce
        move.w  d7,$9c(a6)              ;azzera eventuali IRQ
```

```
        move.w  d6,$96(a6)              ;spegni DMA della voce
        move.w  d7,$9a(a6)              ;spegni INT della voce
        sub.l   a1,a1                           ;FAST CLEAR An
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)      ;68010+ ?
        beq.s   .no010
        lea     .getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010: cmp.l   #lv4irq,$70(a1)
        beq.s   .nochg
        move.l  $70(a1),oldlv4          ;salva l'autovettore del livello 4
        move.l  #lv4irq,$70(a1)         ;imposta il nuovo autovettore
.NoChg: lsl.w   #4-1,d3                 ;d3=d3*8: ora esprime offset di 16 byte
        lea     $dff0a0,a6
        move.w  d2,$8(a6,d3.w)          ;imposta AUDxVOL
        move.l  #clock,d2
        divu.w  d1,d2                   ;d2.w=clock/freq = periodo di camp.
        move.w  d2,$6(a6,d3.w)          ;imposta AUDxPER
        lea     $dff000,a6
        or.w    #$8000,d7
        move.w  d7,$9a(a6)              ;accende INT della voce
        lea     plsregs(pc,d3.w),a1
        movem.l d0/a0,(a1)              ;registri fissi
        movem.l d0/a0,4*2(a1)           ;registri di lavoro
        move.w  d7,$9c(a6)              ;forza IRQ della voce...
        movem.l (sp)+,d0-d7/a0-a1/a6
        rts
.GetVBR:
        dc.l    $4e7a9801       ;movec  vbr,a1  ;base dei vettori di eccezione
        rte
;---------------------------------------
PLSRegs:        ;DEVONO STARE TRA _INIT E _IRQ PER IL MODO DI INDIRIZZAMENTO
                ;USATO: XX(pc,Rn) CHE CONSENTE SOLO 8 BIT CON SEGNO AD "XX"
PLSAud0Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
PLSAud1Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
PLSAud2Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
PLSAud3Regs:    dc.l    0,0     ;lunghezza,puntatore - fissi
                dc.l    0,0     ;lunghezza,puntatore - variabili
;---------------------------------------
PlayLongSample_IRQ:
                ;[a1=PLSAudxRegs]
                ;[d3.w=voce]
        movem.l d0-d3/a0-a1/a6,-(sp)
        and.w   #3,d3                   ;al massimo 3 voci
        move.w  d3,d2
        lsl.w   #4,d3                   ;d3=d3*16: esprime offset di 16 byte
        lea     plsregs(pc,d3.w),a1
        movem.l 4*2(a1),d0/a0           ;grabba i registri di lavoro
        lea     $dff0a0,a6
        move.l  a0,$0(a6,d3.w)          ;imposta AUDxLC
        move.l  d0,d1                   ;d1.l=lunghezza mancante
        and.l   #~(128*1024-1),d1       ;mancano ancora piu' di 128 kB
        bne.s   .long                   ;se SI: vai a .long
        move.l  d0,d1                   ;se NO: usa lungh. mancante (< 128 kB)
.Long:  lsr.l   #1,d1                   ;trasforma la lungh. da suonare in WORD
        move.w  d1,$4(a6,d3.w)          ;imposta AUDxLEN
        add.l   #128*1024,a0            ;punta a0 al prossimo blocco
        sub.l   #128*1024,d0            ;lunghezza MENO 128 kB
        bhi.s   .noloop                 ;d0 => 1 ? (manca ancora ALMENO 1 byte)
```

```
        movem.l (a1),d0/a0              ;se NO: reimposta registri originali
.NoLoop:movem.l d0/a0,4*2(a1)           ;salva comunque d0 e a0 nelle copie
        move.w  #%1<<7,d0
        lsl.w   d2,d0
        move.w  d0,$dff09c              ;azzera IRQ della voce per non subire
                                        ;un nuovo interrupt appena uscito
        moveq   #%1,d0
        lsl.w   d2,d0
        or.w    #$8200,d0               ;accende DMA della voce
        move.w  d0,$dff096
        movem.l (sp)+,d0-d3/a0-a1/a6
        rts
;-------------------------------------
PlayLongSample_restore:
                ;[d3.w=voce (0..3)]
        movem.l d0-d1/d3/a0/a6,-(sp)
        and.w   #3,d3                   ;al massimo 3 voci
        lea     $dff000,a6
        moveq   #1,d0
        lsl.w   d3,d0
        move.w  #1<<7,d1
        lsl.w   d3,d1
        move.w  d1,$9c(a6)              ;azzera eventuali IRQ della voce
        move.w  d0,$96(a6)              ;spegne DMA della voce
        move.w  d1,$9a(a6)              ;spegne INT della voce
        move.w  $1c(a6),d0
        and.w   #$0780,d0               ;spente tutte le voci = ultima voce ?
        bne.s   .NoOFF
        sub.l   a0,a0                   ;se SI:...
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)
        beq.s   .no010
        lea     .getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010: move.l  oldlv4(pc),$70(a0)      ;...reimposta il vecchio autovettore
.NoOFF: lea     $dff000,a6
        add.w   d3,d3                   ;d3=d3*2: esprime offset di word
        move.w  oldints(pc,d3.w),d0
        or.w    #$8000,d0
        move.w  d0,$9a(a6)              ;accende vecchi INT
        move.w  olddmas(pc,d3.w),d0
        or.w    #$8000,d0
        move.w  d0,$96(a6)              ;accende vecchi DMA
        movem.l (sp)+,d0-d1/d3/a0/a6
        rts
.GetVBR:
        dc.l    $4e7a8801       ;movec  vbr,a0  ;base dei vettori di eccezione
        rte
;-------------------------------------
OldINTs:dc.w    0,0,0,0
OldDMAs:dc.w    0,0,0,0
OldLv4: dc.l    0


*************************************
*****  Level 4 Interrupt Handler  *****
*************************************

        cnop    0,8
Lv4IRQ:
        move.w  d3,-(sp)
        pea     .exit(pc)               ;pusha il ritorno per l'RTS nello stack
```

```
        moveq   #3,d3
        btst    #10-8,$dff01e           ;aud3 IRQ ?
        bne.w   playlongsample_irq      ;se SI: bracha (seza ritorno) alla _IRQ

        moveq   #2,d3
        btst    #9-8,$dff01e            ;aud2 IRQ ?
        bne.w   playlongsample_irq

        moveq   #1,d3
        btst    #8-8,$dff01e            ;aud1 IRQ ?
        bne.w   playlongsample_irq

        moveq   #0,d3                   ;aud0 IRQ ?
        btst    #7,$dff01f
        bne.w   playlongsample_irq

.Exit:  move.w  (sp)+,d3                ;anche ritorno per l'RTS della _IRQ
        rte



        SECTION Sample,DATA_C

        ; MammaGamma by Alan Parsons Project (©1981)
Sample:
        incbin  "assembler2:sorgenti8/Mammagamma.17897"
Sample_end:

        END
```

Non c'è molto da dire... Non è real surround, ma gli assomiglia... Provate
a ritardare di più le due voci con un loop o qualcosa del genere e sentite
che effetti fà (arriva a fare l'effetto "sega elettrica" con un ritardo alto).
...Occhio a non ritardare troppo: genereresti un'eco...

28.6 Lezione14-6a

```
; Corso Asm - LEZIONE xx:  ** SUONA SAMPLE MOLTO LUNGHI ANCHE IN FAST **


        SECTION PlayLongSamples,CODE

Start:
        bset    #1,$bfe001              ;spegne il filtro passa-basso
                                        ;>>>> PARAMETRI <<<<
        lea     sample,a0               ;indirizzo sample
        move.l  #sample_end-sample,d0   ;lunghezza sample in byte
        move.w  #17897,d1               ;frequenza di lettura
        moveq   #64,d2                  ;volume
        bsr.s   playlongsample_init     ;INIT routine (comincia)....
                                        ;....CPU libera....
WLMB:   btst    #6,$bfe001              ;testa LMB+RMB...provate, dunque,
        bne.s   wlmb                    ;a girare per il Wb e noterete
        btst    #10,$dff016             ;come non avvenga NESSUN rallentamento
        bne.s   wlmb                    ;....magie del DMA !

        bsr.w   playlongsample_restore  ;RESTORE routine (spegne tutto)
```

```
        rts


*************************************
*****  Play Long Sample Routines  *****
*************************************

PlayLongSample_init:
                ;[a0=sample adr]
                ;[d0.l=lunghezza.b sample, d1.w=frequenza, d2.w=volume]
                ;* L'AutoVector Lv4 IRQ deve essere disponibile *

_LVOSupervisor  equ     -30
_LVOAllocMem    EQU     -198
_LVOFreeMem     EQU     -210
_LVOAvailMem    EQU     -216
MEMF_CHIP       equ     1<<1
MEMF_LARGEST    equ     1<<17
MEMF_CLEAR      equ     1<<16
Clock           equ     3546895
AFB_68010       equ     0
AttnFlags       equ     296

        movem.l d0-d2/a0-a1/a5-a6,-(sp) ;salva molti registri perche' le
                                        ;library sporcano d0-d2/a0-a1
        lea     plsregs(pc),a5
        movem.l d0/a0,(a5)              ;registri fissi di riferimento
        movem.l d0/a0,4*2(a5)           ;registri di lavoro
        move.l  4.w,a6
        move.l  #MEMF_CHIP!MEMF_LARGEST,d1
        jsr     _LVOAvailMem(a6)        ;-> d0.l=blocco di chip di grande
        cmp.l   #2*128*1024,d0          ;d0.l > 256 kB ?
        bls.s   .okmem                  ;se NO: prendi la lungh. del blocco
        move.l  #2*128*1024,d0          ;se SI: bastano 256 kB
.OkMem: and.w   #~%111,d0               ;d0.l=lungh.totale allineata a 64 bit
        move.l  d0,4*4(a5)
        move.l  #MEMF_CHIP!MEMF_CLEAR,d1;MEMF_CLEAR: a 0 la RAM alloccata
        jsr     _LVOAllocMem(a6)        ;allocca 2 banchi da 128 kB adiacenti
        tst.l   d0                      ;d0.l=0 ?
        beq.w   .bye                    ;se SI: RAM non sufficiente -> esci
        move.l  d0,4*5(a5)              ;salva base del PRIMO banco in chip
        move.l  4*4(a5),d1
        lsr.l   #1,d1
        add.l   d1,d0
        move.l  d0,4*6(a5)              ;salve base del SECONDO banco in chip
        movem.l 4(sp),d1-d2             ;ripristina d1-d2 dallo stack
        sub.l   a0,a0
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)      ;68010+ ?
        beq.s   .no010
        lea     getvbr(pc),a5           ;va a routine con comandi privilegiati
        jsr     _LVOSupervisor(a6)      ;in modo supervisore con l'exec
.No010: lea     $dff000,a6
        move.w  #$0780,$9c(a6)          ;azzera eventuali richieste di IRQ
        move.w  $1c(a6),oldint          ;salva INTENA dell'OS
        move.w  #$0780,$9a(a6)          ;maschera INT AUD0-AUD3
        move.l  $70(a0),oldlv4          ;salva l'autovettore del livello 4
        move.l  #lv4irq,$70(a0)         ;imposta il nuovo autovettore
        move.w  d2,$a8(a6)              ;imposta AUD0VOL
        move.w  d2,$b8(a6)              ;imposta AUD1VOL
        move.w  d2,$c8(a6)              ;imposta AUD2VOL
        move.w  d2,$d8(a6)              ;imposta AUD3VOL
```

```
        move.l  #clock,d2
        divu.w  d1,d2                   ;d2.w=clock/freq = periodo di camp.
        move.w  d2,$a6(a6)              ;imposta AUD0PER
        move.w  d2,$b6(a6)              ;imposta AUD1PER
        move.w  d2,$c6(a6)              ;imposta AUD2PER
        move.w  d2,$d6(a6)              ;imposta AUD3PER
        move.w  $2(a6),olddma           ;salva DMACON dell'OS
        move.w  #$c400,$9a(a6)          ;accende AUD3 IRQ - basta lui...
        move.w  #$8400,$9c(a6)          ;forza l'IRQ per comiciare...
        movem.l (sp)+,d0-d2/a0-a1/a5-a6
.Bye:   rts
;---------------------------------------
GetVBR:
        dc.l    $4e7a8801       ;movec  vbr,a0  ;base dei vettori di eccezione
        rte
;---------------------------------------
PlayLongSample_restore:
        movem.l d0-d2/a0-a1/a5-a6,-(sp)
        sub.l   a0,a0
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)
        beq.s   .no010
        lea     getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010: lea     $dff000,a6
        move.w  #$0780,$9c(a6)          ;azzera le richieste di tutti i canali
        move.w  #$0400,$9a(a6)          ;maschera l'INT AUD3
        move.l  oldlv4(pc),$70(a0)      ;reimposta l'autovettore 4 dell'OS
        move.w  #$000f,$96(a6)          ;spegne tutti i DMA audio
        move.w  oldint(pc),d0
        or.w    #$8000,d0               ;imposta SET/CLR che e' a 0 in INTENAR
        move.w  d0,$9a(a6)              ;reimposta l'INTENA dell'OS
        move.w  olddma(pc),d0
        or.w    #$8000,d0               ;imposta SET/CLR che e' a 0 in DMACONR
        move.w  d0,$96(a6)              ;reimposta il DMACON dell'OS
        move.l  4.w,a6
        movem.l plsregs+4*4(pc),d0/a0-a1
        cmp.l   a0,a1                   ;a1 < a0 ? (a1 punta al banco con
        blo.s   .min                    ;indirizzo minore da cui comincia
        move.l  a0,a1                   ;la memoria alloccata ?)
.Min:   jsr     _LVOFreeMem(a6)         ;restituisci la RAM al sistema
        movem.l (sp)+,d0-d2/a0-a1/a5-a6
        rts
;---------------------------------------
PlayLongSample_IRQ:
        movem.l d0-d2/a0-a1/a5-a6,-(sp)
        lea     $dff000,a6
        lea     plsregs+4*4(pc),a5
        movem.l -4*2(a5),d0/a0          ;d0.l=lungh.mancante/a0=base sample
        movem.l (a5),d1/a1              ;d1.l=lungh.banco/a1=base banco
        move.l  a1,$a0(a6)              ;imposta gli AUDLC
        move.l  a1,$b0(a6)
        move.l  a1,$c0(a6)
        move.l  a1,$d0(a6)
        lsr.l   #1,d1                   ;meta' banco
        cmp.l   d0,d1                   ;meta' banco <= lungh.mancante ?
        bls.s   .longc
        move.l  d0,d1                   ;se NO: copia e suona lungh.mancante
.LongC: move.l  d1,d2
        lsr.l   #1,d1                   ;devidi per 2 per AUDLEN in word
        move.w  d1,$a4(a6)              ;imposta gli AUDLEN
        move.w  d1,$b4(a6)
```

```
        move.w  d1,$c4(a6)
        move.w  d1,$d4(a6)
        lsr.l   #1,d1                   ;dividi per 2 per copiare longword
        subq.w  #1,d1
        move.w  #$007,$180(a6)  ;blu quando comicia a copiare
.CopyLp:move.l  (a0)+,(a1)+
        dbra    d1,.copylp
        move.w  #$000,$180(a6)  ;nero quando finisce
        move.l  -4*1(a5),a0
        add.l   d2,a0                   ;punta a0 al prossimo blocco
        sub.l   d2,d0                   ;lunghezza MENO lungh.suonata
        bhi.s   .noloop                 ;d0 => 1 ? (manca ancora ALMENO 1 byte)
        movem.l plsregs(pc),d0/a0       ;se NO: reimposta registri originali
.NoLoop:movem.l d0/a0,-4*2(a5)          ;salva comunque d0 e a0 nelle copie
        movem.l 4*1(a5),a0/a1           ;scambia puntatori ai 2 banchi
        exg     a0,a1           ;commendola viene usato un solo buffer
        movem.l a0/a1,4*1(a5)
        move.w  #$820f,$96(a6)
        movem.l (sp)+,d0-d2/a0-a1/a5-a6
        rts
;---------------------------------------
OldINT: dc.w    0
OldDMA: dc.w    0
OldLv4: dc.l    0
PLSRegs:dc.l    0,0     ;lunghezza,puntatore del sample - fissi
        dc.l    0,0     ;lunghezza,puntatore del sample- variabili
        dc.l    0,0,0   ;lunghezza,puntatore banco 1,puntatore banco 2 - fissi


**************************************
*****  Level 4 Interrupt Handler  *****
**************************************

        cnop    0,8
Lv4IRQ:
        btst    #10-8,$dff01e           ;IRQ AUD3 ?
        beq.s   .exit
        move.w  #$0780,$dff09c
        bsr.w   playlongsample_irq
.Exit:  rte



        SECTION Sample,DATA_F

        ; MammaGamma by Alan Parsons Project (©1981)
Sample:
        incbin  "assembler2:sorgenti8/Mammagamma.17897"
Sample_end:

        END
```

Con queta sezione 6 di sorgenti sull'audio dell'Amiga siamo passati al sofisticato: con questo sorgente (o, se preferite affidare gli handler di interrupt all'exec, modificatelo VOI come nel sorgente 5b in modo da usare il SetIntVector - non e' necessario, in linea di massima: l'OS non usa gli interrupt audio, infatti non ha server chain di livello 4) potete praticamente suoanare qualsiasi cosa che abbiate in memoria ovunque si trovi (a patto che occupi un solo blocco continuo di RAM; fare un player di sample "spezzettati" in vari chunk in giro per le RAM non sarebbe troppo difficile: sarebbe sufficiente usare questo stesso sorgente in modo che legga sample diversi in vari punti; l'unico problema sarebbe includere un file spezzandolo

```
- cosa che l'assemblatore NON fa - con le routine della DOS library che
leggono porzioni di file: a questo punto, fatta la routine di LOAD, avete
anche fatto un ottimo player da CLI !) siete finalmente in grado di suonare
un sample situato ovunque in memoria, nel chunk più grande che l'AllocMem
riesce a trovare (MEMF_ANY).

Il funzionamento della routine è estremamente semplice: dato un sample di
lunghezza indeterminata in un blocco di RAM QUALSIASI (chip o fast), viene
alloccato un blocco di chip RAM (MEMF_CHIP) di 256 kB - se possibile - o meno,
che viene suddiviso in due buffer da 128 kB - o meno - l'uno in cui copiare
con un loop di CPU i dati del sample di 128 kB - o meno - in 128 kB - o meno -,
al fine di riuscire a far leggere il DMA.
Il motivo dell'uso dei 2 buffer è molto semplice: mentre l'audio ne suona uno,
la CPU ne riempie un'altro con i dati successivi a quello in fase di lettura.

N.B.:   per la verità, certe CPU come il 68040 od il 68030 sono talmente veloci
        da riuscire a copiare tutto il blocco di 128 kB - o meno - in poco più
        di un raster; per cui, anche se non usate due buffer, soprattutto,
        quando il buffer è molto piccolo, è onestamente impossibile sentire
        il DMA suonare gli stessi dati 2 volte nello stesso buffer che loopa,
        perchè la CPU li ha già copiati quando stanno ancora venendo lette
        le prime word.
        I motivi per cui sono stati utilizzati due buffer separati sono
        i seguenti: innanzitutto, per eleganza di coding: IN TEORIA i due
        buffer sono necessari; inoltre, su CPU lente come il 68000 a 16 bit
        di accesso alla RAM dell'Amiga 500, la copia non è poi così istantanea;
        infine, così com'è, la routine avrebbe un bug: l'ultimo blocco del
        sample verrebbe suonato 2 volte, prima di loopare (per allenamento,
        cercate di capire perchè ed aggiustate la ruotine...).

        La lunghezza minima per i buffer è di 4 byte ciascuno; provate
        ad allocare solo 8 byte in tutto e suonare un sample alla frequenza
        di lettura massima (28000 Hz ca., periodo=123): ebbene si, lo 040
        - non si sa bene come - riesce a tener testa al DMA anche con
        2 buffer di un longword !!! Provare per credere...

P.S.:   sulla _IRQ trovate 2 linee commentate: servono a cambiare il colore
        di sfondo ogni volta che, chiamato l'interrupt, la CPU comincia
        a copiare i dati dalla RAM sorgente ai buffer: togliete i commenti
        per renedervi conto di cosa stà combinando il processore mentre
        il DMA suona ignaro del cambiamento di dati...
```

Lezione14-6b

```
; Corso Asm - LEZIONE xx:  ** SUONA SAMPLE MOLTO LUNGHI ANCHE IN FAST 2 **


        SECTION PlayLongSamples,CODE

Start:
        bset    #1,$bfe001              ;spegne il filtro passa-basso
                                        ;>>>> PARAMETRI <<<<
        lea     sample,a0               ;indirizzo sample
        move.l  #sample_end-sample,d0   ;lunghezza sample in byte
        move.w  #17897,d1               ;frequenza di lettura
        moveq   #64,d2                  ;volume
        bsr.s   playlongsample_init     ;INIT routine (comincia)....
                                        ;....CPU libera....
WLMB:   btst    #6,$bfe001              ;testa LMB+RMB...provate, dunque,
        bne.s   wlmb                    ;a girare per il Wb e noterete
        btst    #10,$dff016             ;come non avvenga NESSUN rallentamento
```

```
        bne.s   wlmb                    ;....magie del DMA !

        bsr.w   playlongsample_restore  ;RESTORE routine (spegne tutto)
        rts


*************************************
*****  Play Long Sample Routines  *****
*************************************

PlayLongSample_init:
                ;[a0=sample adr]
                ;[d0.l=lunghezza.b sample, d1.w=frequenza, d2.w=volume]
                ;* L'AutoVector Lv4 IRQ deve essere disponibile *

_LVOSupervisor  equ     -30
_LVOAllocMem    EQU     -198
_LVOFreeMem     EQU     -210
_LVOAvailMem    EQU     -216
MEMF_CHIP       equ     1<<1
MEMF_LARGEST    equ     1<<17
MEMF_CLEAR      equ     1<<16
AFB_68010       equ     0
AttnFlags       equ     296

Clock           equ     3546895
MaxBanco1       equ     32*1024
MaxBanco2       equ     32*1024

        movem.l d0-d2/a0-a1/a5-a6,-(sp)
        lea     plsregs(pc),a5
        movem.l d0/a0,(a5)              ;registri fissi di riferimento
        movem.l d0/a0,4*2(a5)           ;registri di lavoro
        move.l  4.w,a6
        move.l  #MEMF_CHIP!MEMF_LARGEST,d1
        jsr     _LVOAvailMem(a6)        ;-> d0.l=blocco di chip di grande
        cmp.l   #maxbanco1,d0           ;d0.l > MaxBanco1 kB ?
        bls.s   .okmem1                 ;se NO: prendi la lungh. del blocco
        move.l  #maxbanco1,d0           ;se SI: bastano MaxBanco1 kB
.OkMem1:and.w   #~%11,d0                ;d0.l=lungh.banco allineata a 32 bit
        move.l  d0,4*4(a5)
        move.l  #MEMF_CHIP!MEMF_CLEAR,d1;MEMF_CLEAR: a 0 la RAM alloccata
        jsr     _LVOAllocMem(a6)        ;allocca 1 banco da MaxBanco1 kB
        tst.l   d0                      ;d0.l=0 ?
        beq.w   .bye                    ;se SI: RAM non sufficiente -> esci
        move.l  d0,4*5(a5)              ;salva base del PRIMO banco in chip
        move.l  #MEMF_CHIP!MEMF_LARGEST,d1
        jsr     _LVOAvailMem(a6)        ;-> d0.l=blocco di chip di grande
        cmp.l   #maxbanco2,d0           ;d0.l > MaxBanco2 kB ?
        bls.s   .okmem2                 ;se NO: prendi la lungh. del blocco
        move.l  #maxbanco2,d0           ;se SI: bastano MaxBanco2 kB
.OkMem2:and.w   #~%11,d0                ;d0.l=lungh.banco allineata a 32 bit
        move.l  d0,4*6(a5)
        move.l  #MEMF_CHIP!MEMF_CLEAR,d1;MEMF_CLEAR: a 0 la RAM alloccata
        jsr     _LVOAllocMem(a6)        ;allocca 1 banco da MaxBanco2 kB
        tst.l   d0                      ;d0.l=0 ?
        beq.w   .bye                    ;se SI: RAM non sufficiente -> esci
        move.l  d0,4*7(a5)              ;salva base del SECONDO banco in chip
        movem.l 4(sp),d1-d2             ;ripristina d1-d2 dallo stack
        sub.l   a0,a0
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)      ;68010+ ?
```

```
        beq.s   .no010
        lea     getvbr(pc),a5           ;va a routine con comandi privilegiati
        jsr     _LVOSupervisor(a6)      ;in modo supervisore con l'exec
.No010: lea     $dff000,a6
        move.w  #$0780,$9c(a6)          ;azzera eventuali richieste di IRQ
        move.w  $1c(a6),oldint          ;salva INTENA dell'OS
        move.w  #$0780,$9a(a6)          ;maschera INT AUD0-AUD3
        move.l  $70(a0),oldlv4          ;salva l'autovettore del livello 4
        move.l  #lv4irq,$70(a0)         ;imposta il nuovo autovettore
        move.w  d2,$a8(a6)              ;imposta AUD0VOL
        move.w  d2,$b8(a6)              ;imposta AUD1VOL
        move.w  d2,$c8(a6)              ;imposta AUD2VOL
        move.w  d2,$d8(a6)              ;imposta AUD3VOL
        move.l  #clock,d2
        divu.w  d1,d2                   ;d2.w=clock/freq = periodo di camp.
        move.w  d2,$a6(a6)              ;imposta AUD0PER
        move.w  d2,$b6(a6)              ;imposta AUD1PER
        move.w  d2,$c6(a6)              ;imposta AUD2PER
        move.w  d2,$d6(a6)              ;imposta AUD3PER
        move.w  $2(a6),olddma           ;salva DMACON dell'OS
        move.w  #$c400,$9a(a6)          ;accende AUD3 IRQ - basta lui...
        move.w  #$8400,$9c(a6)          ;forza l'IRQ per comiciare...
        movem.l (sp)+,d0-d2/a0-a1/a5-a6
.Bye:   rts
;---------------------------------------
GetVBR:
        dc.l    $4e7a8801       ;movec  vbr,a0  ;base dei vettori di eccezione
        rte
;---------------------------------------
PlayLongSample_restore:
        movem.l d0-d2/a0-a1/a5-a6,-(sp)
        sub.l   a0,a0
        move.l  4.w,a6
        btst    #afb_68010,attnflags+1(a6)
        beq.s   .no010
        lea     getvbr(pc),a5
        jsr     _LVOSupervisor(a6)
.No010: lea     $dff000,a6
        move.w  #$0780,$9c(a6)          ;azzera le richieste di tutti i canali
        move.w  #$0400,$9a(a6)          ;maschera l'INT AUD3
        move.l  oldlv4(pc),$70(a0)      ;reimposta l'autovettore 4 dell'OS
        move.w  #$000f,$96(a6)          ;spegne tutti i DMA audio
        move.w  oldint(pc),d0
        or.w    #$8000,d0               ;imposta SET/CLR che e' a 0 in INTENAR
        move.w  d0,$9a(a6)              ;reimposta l'INTENA dell'OS
        move.w  olddma(pc),d0
        or.w    #$8000,d0               ;imposta SET/CLR che e' a 0 in DMACONR
        move.w  d0,$96(a6)              ;reimposta il DMACON dell'OS
        move.l  4.w,a6
        movem.l plsregs+4*4(pc),d0/a1
        jsr     _LVOFreeMem(a6)         ;libera la RAM del PRIMO banco
        movem.l plsregs+4*6(pc),d0/a1
        jsr     _LVOFreeMem(a6)         ;libera la RAM del SECONDO banco
        movem.l (sp)+,d0-d2/a0-a1/a5-a6
        rts
;---------------------------------------
PlayLongSample_IRQ:
        movem.l d0-d2/a0-a1/a5-a6,-(sp)
        lea     $dff000,a6
        lea     plsregs(pc),a5
        movem.l 4*2(a5),d0/a0           ;d0.l=lungh.mancante/a0=base sample
        movem.l 4*4(a5),d1/a1           ;d1.l=lungh.banco/a1=base banco
```

```
        move.l  a1,$a0(a6)              ;imposta gli AUDLC
        move.l  a1,$b0(a6)
        move.l  a1,$c0(a6)
        move.l  a1,$d0(a6)
        cmp.l   d0,d1                   ;banco <= lungh.mancante ?
        bls.s   .longc
        move.l  d0,d1                   ;se NO: copia e suona lungh.mancante
.LongC: move.l  d1,d2
        lsr.l   #1,d1                   ;devidi per 2 per AUDLEN in word
        move.w  d1,$a4(a6)              ;imposta gli AUDLEN
        move.w  d1,$b4(a6)
        move.w  d1,$c4(a6)
        move.w  d1,$d4(a6)
        lsr.l   #1,d1                   ;dividi per 2 per copiare longword
        subq.w  #1,d1
        move.w  #$007,$180(a6)  ;blu quando comicia a copiare
.CopyLp:move.l  (a0)+,(a1)+
        dbra    d1,.copylp
        move.w  #$000,$180(a6)  ;nero quando finisce
        move.l  4*3(a5),a0
        add.l   d2,a0                   ;punta a0 al prossimo blocco
        sub.l   d2,d0                   ;lunghezza MENO lungh.suonata
        bhi.s   .noloop                 ;d0 => 1 ? (manca ancora ALMENO 1 byte)
        movem.l (a5),d0/a0              ;se NO: reimposta registri originali
.NoLoop:movem.l d0/a0,4*2(a5)           ;salva comunque d0 e a0 nelle copie
        movem.l 4*4(a5),d0-d1/a0-a1     ;scambia puntatori e lunghezza
        exg     d0,a0           ;commendole viene usato un solo buffer
        exg     d1,a1           ;
        movem.l d0-d1/a0-a1,4*4(a5)
        move.w  #$820f,$96(a6)
        movem.l (sp)+,d0-d2/a0-a1/a5-a6
        rts
;---------------------------------------
OldINT: dc.w    0
OldDMA: dc.w    0
OldLv4: dc.l    0
PLSRegs:dc.l    0,0     ;lunghezza,puntatore del sample - fissi
        dc.l    0,0     ;lunghezza,puntatore del sample- variabili
        dc.l    0,0     ;lunghezza,puntatore del banco 1 - fissi
        dc.l    0,0     ;lunghezza,puntatore del banco 2 - fissi


*************************************
*****  Level 4 Interrupt Handler  *****
*************************************

        cnop    0,8
Lv4IRQ:
        btst    #10-8,$dff01e           ;IRQ AUD3 ?
        beq.s   .exit
        move.w  #$0780,$dff09c
        bsr.w   playlongsample_irq
.Exit:  rte



        SECTION Sample,DATA_F

        ; MammaGamma by Alan Parsons Project (©1981)
Sample:
        incbin  "assembler2:sorgenti8/Mammagamma.17897"
Sample_end:
```

```
        END


Questa volta non è cambiato poi molto: ora i banchi sono di lunghezza e di
posizione indipendente, non più adiacenti ne' lunghi uguali.
La routine non ha subito particolari cambiamenti: allocca i due buffer
separatamente, e nella _IRQ si limita a scambiare anche le loro lunghezze,
oltre ai puntatori.

N.B.:   Le note dell'esempio precedente valgono anche per questo.
```

28.7 Lezione14-7a1

```
f9fa f9fa 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0a3b 2043
6f72 736f 2041 736d 202d 204c 455a 494f
4e45 2078 783a 2020 2a2a 204d 4f44 554c
4152 4520 494e 2041 4d50 4945 5a5a 4120
554e 2741 524d 4f4e 4943 4120 2a2a 0a0a
0953 4543 5449 4f4e 094c 455a 494f 4e45
7878 312c 434f 4445 0a0a 5374 6172 743a
0a0a 096c 6561 096d 6f64 766f 6c2c 6130
0a09 6d6f 7665 7109 2330 2c64 300a 096d
6f76 6571 0923 3635 2d31 2c64 370a 2e4c
7031 3a09 6d6f 7665 2e77 0964 302c 2861
3029 2b0a 0961 6464 712e 7709 2331 2c64
300a 0964 6272 6109 6437 2c2e 6c70 310a
0973 7562 712e 7709 2331 2c64 300a 2e4c
7032 3a09 6d6f 7665 2e77 0964 302c 2861
3029 2b0a 0964 6272 6109 6430 2c2e 6c70
320a 0a5f 4c56 4f44 6973 6162 6c65 0945
5155 092d 3132 300a 5f4c 564f 456e 6162
6c65 0945 5155 092d 3132 360a 0a09 6d6f
7665 2e6c 0934 2e77 2c61 360a 096a 7372
095f 4c56 4f44 6973 6162 6c65 2861 3629
0a0a 0962 7365 7409 2331 2c24 6266 6530
3031 0909 3b73 7065 676e 6520 696c 2066
696c 7472 6f20 7061 7373 612d 6261 7373
6f0a 0a09 6c65 6109 2464 6666 3030 302c
6136 0a09 6d6f 7665 2e77 0924 3228 6136
292c 6437 0909 3b73 616c 7661 2044 4d41
2064 656c 6c27 4f53 0a09 6d6f 7665 2e77
0924 3130 2861 3629 2c64 3609 093b 7361
6c76 6120 4144 4b43 4f4e 2064 656c 6c27
4f53 0a0a 436c 6f63 6b09 6571 7509 3335
3436 3839 350a 0a09 6d6f 7665 2e6c 0923
6172 6d6f 6e69 6361 2c24 6230 2861 3629
0a09 6d6f 7665 2e77 0923 3136 2f32 2c24
6234 2861 3629 0a09 6d6f 7665 2e77 0923
636c 6f63 6b2f 2831 362a 3838 3029 2c24
6236 2861 3629 0a0a 096d 6f76 652e 6c09
236d 6f64 766f 6c2c 2461 3028 6136 290a
096d 6f76 652e 7709 2328 6d6f 6476 6f6c
5f65 6e64 2d6d 6f64 766f 6c29 2f32 2c24
6134 2861 3629 0a09 6d6f 7665 2e77 0923
636c 6f63 6b2f 286d 6f64 766f 6c5f 656e
642d 6d6f 6476 6f6c 292c 2461 3628 6136
290a 0a09 6d6f 7665 2e77 0923 2438 3030
312c 2439 6528 6136 2909 093b 696d 706f
```

```
7374 6120 5553 4530 5631 0a0a 096d 6f76
652e 7709 2324 3832 3033 2c24 3936 2861
3629 0909 3b61 6363 656e 6465 2041 5544
3020 6520 4155 4431 2069 6e20 444d 4143
4f4e 570a 0a57 4c4d 423a 0962 7473 7409
2336 2c24 6266 6530 3031 0909 3b61 7370
6574 7461 2069 6c20 7075 6c73 616e 7465
2073 696e 6973 7472 6f20 6465 6c20 6d6f
7573 650a 0962 6e65 2e73 0957 4c4d 420a
0a09 6d6f 7665 2e77 0923 2430 3030 312c
2439 3628 6136 2909 093b 7370 6567 6e65
2055 5345 3056 310a 096f 722e 7709 2324
3830 3030 2c64 3609 093b 6163 6365 6e64
6520 696c 2062 6974 2031 3520 2853 4554
2f43 4c52 290a 096d 6f76 652e 7709 6436
2c24 3965 2861 3629 0909 3b72 6569 6d70
6f73 7461 2041 444b 434f 4e20 6465 6c6c
274f 530a 096d 6f76 652e 7709 2324 3030
3033 2c24 3936 2861 3629 0909 3b73 7065
676e 6520 4155 4430 2065 2041 5544 310a
096f 722e 7709 2324 3830 3030 2c64 3709
093b 6163 6365 6e64 6520 696c 2062 6974
2031 3520 2853 4554 2f43 4c52 290a 096d
6f76 652e 7709 6437 2c24 3936 2861 3629
0909 3b72 6569 6d70 6f73 7461 2044 4d41
2064 656c 6c27 4f53 0a09 6d6f 7665 2e6c
0934 2e77 2c61 360a 096a 7372 095f 4c56
4f45 6e61 626c 6528 6136 290a 0972 7473
0a0a 0953 4543 5449 4f4e 0953 616d 706c
652c 4441 5441 5f43 093b 7665 6e65 6e64
6f20 6c65 7474 6120 6461 6c20 444d 4120
6465 7665 2065 7373 6572 6520 696e 2043
4849 500a 0a41 726d 6f6e 6963 613a 093b
6172 6d6f 6e69 6361 2064 6920 3136 2076
616c 6f72 6920 6372 6561 7461 2063 6f6c
2749 4320 6465 6c20 7472 6173 6827 6d2d
6f6e 650a 0944 432e 4209 2431 392c 2434
362c 2436 392c 2437 432c 2437 442c 2436
412c 2434 372c 2431 412c 2445 382c 2442
422c 2439 372c 2438 342c 2438 332c 2439
352c 2442 382c 2445 350a 4d6f 6456 6f6c
3a0a 0962 6c6b 2e77 0936 352a 320a 4d6f
6456 6f6c 5f65 6e64 3a0a 0945 4e44 0a0a
0a4d 6f6c 746f 2073 656d 706c 6963 656d
656e 7465 2c20 6162 6269 616d 6f20 696e
6e61 6e7a 6974 7574 746f 2067 656e 6572
6174 6f20 756e 6120 7461 6265 6c6c 6120
6469 2031 3330 2076 616c 6f72 690a 6461
2030 2061 2036 3420 6520 6461 2036 3420
6120 3020 7065 7220 6920 766f 6c75 6d69
2064 656c 6c27 4155 4431 564f 4c2c 2063
6865 2061 6262 6961 6d6f 2066 6174 746f
206c 6567 6765 7265 2061 6c0a 6361 6e61
6c65 2030 2c20 6d65 6e74 7265 2069 6c20
6361 6e61 6c65 2031 206c 6567 6765 7661
206c 2761 726d 6f6e 6963 6120 616c 6c61
2066 7265 7175 656e 7a61 2064 656c 204c
4133 2028 3838 3020 487a 0a64 6920 6672
6571 7565 6e7a 6120 6427 6f6e 6461 292e
0a43 6f6d 6520 7065 7269 6f64 6f20 6465
6c20 6361 6e61 6c65 206d 6f64 756c 6174
6f72 6520 6162 6269 616d 6f20 6661 7474
```

```
6f20 6669 6e74 6120 6368 6520 7374 6573
7365 206c 6567 6765 6e64 6f20 756e 0a6e
6f72 6d61 6c65 2073 616d 706c 6520 6520
676c 6920 6162 6269 616d 6f20 666f 726e
6974 6f20 6c61 2076 656c 6f63 6974 e020
6469 206c 6574 7475 7261 3a20 7065 7263
68e8 206c 6120 7461 6265 6c6c 610a 7665
6e67 6120 7475 7474 6120 6c65 7474 6120
696e 2031 2073 6563 6f6e 646f 2069 6c20
7065 7269 6f64 6f20 6469 2063 616d 7069
6f6e 616d 656e 746f 2064 6576 6520 6573
7365 7265 2070 6172 690a 616c 6c61 2063
6f73 7461 6e74 6520 6469 2063 6c6f 636b
2064 6976 6973 6120 7065 7220 6c61 206c
756e 6768 657a 7a61 2069 6e20 6279 7465
2064 656c 6c61 2074 6162 656c 6c61 203d
2031 2048 7a2e 0a0a 4e2e 422e 3a09 6e6f
7461 7465 2063 6865 206e 6f6e 20e8 2073
7461 746f 2069 6d70 6f73 7461 746f 2069
6c20 766f 6c75 6d65 2064 656c 2063 616e
616c 6520 3020 2841 5544 3056 4f4c 292c
0a09 706f 6963 68e8 206e 6f6e 20e8 206e
6563 6365 7373 6172 696f 2c20 696e 2071
7561 6e74 6f20 696c 2073 756f 206f 7574
7075 7420 6e6f 6e20 7669 656e 650a 0964
6561 6d70 6c69 6669 6361 746f 2028 3634
203d 202d 3020 6442 2920 6520 6669 6e69
7363 6520 6469 7265 7474 616d 656e 7465
206e 656c 2072 6567 6973 7472 6f20 4155
4431 564f 4c2e 0a09 4e65 6d6d 656e 6f20
4155 4431 564f 4c20 e820 7374 6174 6f20
696d 706f 7374 6174 6f20 616c 6c27 696e
697a 696f 2c20 706f 6963 68e8 2076 6965
6e65 2073 7562 6974 6f0a 096d 6f64 6966
6963 6174 6f20 6461 6c20 6d6f 6475 6c61
746f 7265 2e0a
```

Lezione14-7a2

```
f9fa f9fa 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0a3b 2043
6f72 736f 2041 736d 202d 204c 455a 494f
4e45 2078 783a 2020 2a2a 204d 4f44 554c
4152 4520 494e 2041 4d50 4945 5a5a 4120
554e 2741 524d 4f4e 4943 4120 494e 2053
5445 5245 4f2a 2a0a 0a09 5345 4354 494f
4e09 4c45 5a49 4f4e 4578 7837 6132 2c43
4f44 450a 0a53 7461 7274 3a0a 0a09 6c65
6109 6d6f 6476 6f6c 312c 6130 0a09 6d6f
7665 7109 2330 2c64 300a 096d 6f76 6571
0923 3635 2d31 2c64 370a 2e4c 7031 3a09
6d6f 7665 2e77 0964 302c 2861 3029 2b0a
0961 6464 712e 7709 2331 2c64 300a 0964
6272 6109 6437 2c2e 6c70 310a 0973 7562
712e 7709 2331 2c64 300a 2e4c 7032 3a09
6d6f 7665 2e77 0964 302c 2861 3029 2b0a
0964 6272 6109 6430 2c2e 6c70 320a 0a09
6c65 6109 6d6f 6476 6f6c 322c 6130 0a09
6d6f 7665 7109 2336 352c 6430 0a09 6d6f
7665 7109 2336 352d 312c 6437 0a2e 4c70
```

```
333a 0973 7562 712e 7709 2331 2c64 300a
096d 6f76 652e 7709 6430 2c28 6130 292b
0a09 6462 7261 0964 372c 2e6c 7033 0a09
6d6f 7665 7109 2336 352d 312c 6437 0a2e
4c70 343a 096d 6f76 652e 7709 6430 2c28
6130 292b 0a09 6164 6471 2e77 0923 312c
6430 0a09 6462 7261 0964 372c 2e6c 7034
0a0a 5f4c 564f 4469 7361 626c 6509 4551
5509 2d31 3230 0a5f 4c56 4f45 6e61 626c
6509 4551 5509 2d31 3236 0a0a 096d 6f76
652e 6c09 342e 772c 6136 0a09 6a73 7209
5f4c 564f 4469 7361 626c 6528 6136 290a
0a09 6273 6574 0923 312c 2462 6665 3030
3109 093b 7370 6567 6e65 2069 6c20 6669
6c74 726f 2070 6173 7361 2d62 6173 736f
0a0a 096c 6561 0924 6466 6630 3030 2c61
360a 096d 6f76 652e 7709 2432 2861 3629
2c64 3709 093b 7361 6c76 6120 444d 4120
6465 6c6c 274f 530a 096d 6f76 652e 7709
2431 3028 6136 292c 6436 0909 3b73 616c
7661 2041 444b 434f 4e20 6465 6c6c 274f
530a 0a43 6c6f 636b 0965 7175 0933 3534
3638 3935 0a0a 096d 6f76 652e 6c09 2361
726d 6f6e 6963 612c 2462 3028 6136 290a
096d 6f76 652e 7709 2331 362f 322c 2462
3428 6136 290a 096d 6f76 652e 7709 2363
6c6f 636b 2f28 3136 2a34 3430 292c 2462
3628 6136 2909 3b4c 4132 0a09 6d6f 7665
2e6c 0923 6172 6d6f 6e69 6361 2c24 6430
2861 3629 0a09 6d6f 7665 2e77 0923 3136
2f32 2c24 6434 2861 3629 0a09 6d6f 7665
2e77 0923 636c 6f63 6b2f 2831 362a 3434
3029 2c24 6436 2861 3629 093b 4c41 320a
0a09 6d6f 7665 2e6c 0923 6d6f 6476 6f6c
312c 2461 3028 6136 290a 096d 6f76 652e
7709 2328 6d6f 6476 6f6c 315f 656e 642d
6d6f 6476 6f6c 3129 2f32 2c24 6134 2861
3629 0a09 6d6f 7665 2e77 0923 636c 6f63
6b2f 2828 6d6f 6476 6f6c 315f 656e 642d
6d6f 6476 6f6c 3129 2f32 292c 2461 3628
6136 290a 096d 6f76 652e 6c09 236d 6f64
766f 6c32 2c24 6330 2861 3629 0a09 6d6f
7665 2e77 0923 286d 6f64 766f 6c32 5f65
6e64 2d6d 6f64 766f 6c32 292f 322c 2463
3428 6136 290a 096d 6f76 652e 7709 2363
6c6f 636b 2f28 286d 6f64 766f 6c32 5f65
6e64 2d6d 6f64 766f 6c32 292f 3229 2c24
6336 2861 3629 0a0a 096d 6f76 652e 7709
2324 3830 3035 2c24 3965 2861 3629 0909
3b69 6d70 6f73 7461 2055 5345 3056 3120
6520 5553 4532 5633 0a0a 096d 6f76 652e
7709 2324 3832 3066 2c24 3936 2861 3629
0909 3b61 6363 656e 6465 2041 5544 302d
4155 4433 2069 6e20 444d 4143 4f4e 570a
0a57 4c4d 423a 0962 7473 7409 2336 2c24
6266 6530 3031 0909 3b61 7370 6574 7461
2069 6c20 7075 6c73 616e 7465 2073 696e
6973 7472 6f20 6465 6c20 6d6f 7573 650a
0962 6e65 2e73 0957 4c4d 420a 0a09 6d6f
7665 2e77 0923 2430 3030 352c 2439 6528
6136 2909 093b 7370 6567 6e65 2055 5345
3056 3120 6520 5553 4532 5633 0a09 6f72
```

```
2e77 0923 2438 3030 302c 6436 0909 3b61
6363 656e 6465 2069 6c20 6269 7420 3135
2028 5345 542f 434c 5229 0a09 6d6f 7665
2e77 0964 362c 2439 6528 6136 2909 093b
7265 696d 706f 7374 6120 4144 4b43 4f4e
2064 656c 6c27 4f53 0a09 6d6f 7665 2e77
0923 2430 3030 662c 2439 3628 6136 2909
093b 7370 6567 6e65 2041 5544 302d 4155
4433 0a09 6f72 2e77 0923 2438 3030 302c
6437 0909 3b61 6363 656e 6465 2069 6c20
6269 7420 3135 2028 5345 542f 434c 5229
0a09 6d6f 7665 2e77 0964 372c 2439 3628
6136 2909 093b 7265 696d 706f 7374 6120
444d 4120 6465 6c6c 274f 530a 096d 6f76
652e 6c09 342e 772c 6136 0a09 6a73 7209
5f4c 564f 456e 6162 6c65 2861 3629 0a09
7274 730a 0a09 5345 4354 494f 4e09 5361
6d70 6c65 2c44 4154 415f 4309 3b76 656e
656e 646f 206c 6574 7461 2064 616c 2044
4d41 2064 6576 6520 6573 7365 7265 2069
6e20 4348 4950 0a0a 4172 6d6f 6e69 6361
3a09 3b61 726d 6f6e 6963 6120 6469 2031
3620 7661 6c6f 7269 2063 7265 6174 6120
636f 6c27 4943 2064 656c 2074 7261 7368
276d 2d6f 6e65 0a09 4443 2e42 0924 3139
2c24 3436 2c24 3639 2c24 3743 2c24 3744
2c24 3641 2c24 3437 2c24 3141 2c24 4538
2c24 4242 2c24 3937 2c24 3834 2c24 3833
2c24 3935 2c24 4238 2c24 4535 0a4d 6f64
566f 6c31 3a0a 0962 6c6b 2e77 0936 352a
320a 4d6f 6456 6f6c 315f 656e 643a 0a4d
6f64 566f 6c32 3a0a 0962 6c6b 2e77 0936
352a 320a 4d6f 6456 6f6c 325f 656e 643a
0a09 454e 440a 0a0a 5175 6573 7461 2076
6f6c 7461 2061 6262 6961 6d6f 2063 7265
6174 6f20 3220 7461 6265 6c6c 6520 2273
6661 7361 7465 223a 206c 6120 7072 696d
612c 206c 6574 7461 2064 616c 2063 616e
616c 6520 302c 0a6d 6f64 756c 6120 696e
2076 6f6c 756d 6520 696c 2063 616e 616c
6520 3120 6461 2030 2061 2036 342c 2065
2064 6120 3634 2061 2030 3b20 6c61 2073
6563 6f6e 6461 2c20 6c65 7474 6120 6461
6c0a 6361 6e61 6c65 2032 2c20 6d6f 6475
6c61 2075 6775 616c 6d65 6e74 6520 696e
2061 6d70 6965 7a7a 6120 696c 2063 616e
616c 6520 3320 6461 2036 3420 6120 302c
2065 2064 6120 3020 6120 3634 2e0a 4369
f220 7072 6f76 6f63 6120 756e 2065 6666
6574 746f 2064 6920 6170 7061 7265 6e74
6520 2264 6563 656e 7472 616d 656e 746f
2220 6465 6c6c 2775 7363 6974 6120 6465
6c20 7375 6f6e 6f0a 696e 2075 6e27 696d
7069 616e 746f 2053 5445 5245 4f2e 0a51
7569 206c 6120 6c65 7474 7572 6120 6465
6c6c 6520 7461 6265 6c6c 6520 6176 7669
656e 6520 616c 6c61 2066 7265 7175 656e
7a61 2064 6920 226d 657a 7a6f 2220 487a
2c20 696e 2071 7561 6e74 6f0a 7669 656e
6520 6c65 7474 6120 6d65 74e0 2074 6162
656c 6c61 2061 6420 3120 487a 2e0a 0a4e
2e42 2e3a 0973 6520 6176 6574 6520 756e
```

```
2769 6d70 6961 6e74 6f20 4d4f 4e4f 2c20
646f 7672 6573 7465 2073 656e 7469 7265
2075 6e61 206e 6f74 6120 636f 6e74 696e
7561 2073 656e 7a61 0a09 616c 6375 6e61
206d 6f64 756c 617a 696f 6e65 2c20 696e
2071 7561 6e74 6f20 616c 2063 616c 6172
6520 6465 6c20 766f 6c75 6d65 2064 6920
756e 6120 6361 7373 610a 096c 2761 6c74
7261 2073 6920 616c 7a61 2065 206e 6520
636f 6d70 656e 7361 206c 276f 7574 7075
742e 0a
```

Lezione14-7b

```
f9fa f9fa 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0a3b 2043
6f72 736f 2041 736d 202d 204c 455a 494f
4e45 2078 783a 2020 2a2a 204d 4f44 554c
4152 4520 494e 2046 5245 5155 454e 5a41
2055 4e27 4152 4d4f 4e49 4341 202a 2a0a
0a09 5345 4354 494f 4e09 4c45 5a49 4f4e
4578 7837 622c 434f 4445 0a0a 5374 6172
743a 0a0a 096c 6561 096d 6f64 6671 2c61
300a 096d 6f76 6571 0923 3132 332c 6430
0a09 6d6f 7665 2e77 0923 3530 302d 312c
6437 0a2e 4c70 313a 096d 6f76 652e 7709
6430 2c28 6130 292b 0a09 6164 6471 2e77
0923 312c 6430 0a09 6462 7261 0964 372c
2e6c 7031 0a09 6d6f 7665 2e77 0923 3530
302d 312c 6437 0a2e 4c70 323a 096d 6f76
652e 7709 6430 2c28 6130 292b 0a09 7375
6271 2e77 0923 312c 6430 0a09 6462 7261
0964 372c 2e6c 7032 0a0a 5f4c 564f 4469
7361 626c 6509 4551 5509 2d31 3230 0a5f
4c56 4f45 6e61 626c 6509 4551 5509 2d31
3236 0a0a 096d 6f76 652e 6c09 342e 772c
6136 0a09 6a73 7209 5f4c 564f 4469 7361
626c 6528 6136 290a 0a09 6273 6574 0923
312c 2462 6665 3030 3109 093b 7370 6567
6e65 2069 6c20 6669 6c74 726f 2070 6173
7361 2d62 6173 736f 0a0a 096c 6561 0924
6466 6630 3030 2c61 360a 096d 6f76 652e
7709 2432 2861 3629 2c64 3709 093b 7361
6c76 6120 444d 4120 6465 6c6c 274f 530a
096d 6f76 652e 7709 2431 3028 6136 292c
6436 0909 3b73 616c 7661 2041 444b 434f
4e20 6465 6c6c 274f 530a 0a43 6c6f 636b
0965 7175 0933 3534 3638 3935 0a0a 096d
6f76 652e 6c09 2361 726d 6f6e 6963 612c
2462 3028 6136 290a 096d 6f76 652e 7709
2331 362f 322c 2462 3428 6136 290a 096d
6f76 652e 7709 2336 342c 2462 3828 6136
290a 0a09 6d6f 7665 2e6c 0923 6d6f 6466
712c 2461 3028 6136 290a 096d 6f76 652e
7709 2328 6d6f 6466 715f 656e 642d 6d6f
6466 7129 2f32 2c24 6134 2861 3629 0a09
6d6f 7665 2e77 0923 636c 6f63 6b2f 2828
6d6f 6466 715f 656e 642d 6d6f 6466 7129
2f32 292c 2461 3628 6136 290a 0a09 6d6f
7665 2e77 0923 2438 3031 302c 2439 6528
```

```
6136 2909 093b 696d 706f 7374 6120 5553
4530 5031 0a0a 096d 6f76 652e 7709 2324
3832 3033 2c24 3936 2861 3629 0909 3b61
6363 656e 6465 2041 5544 3020 6520 4155
4431 2069 6e20 444d 4143 4f4e 570a 0a57
4c4d 423a 0962 7473 7409 2336 2c24 6266
6530 3031 0909 3b61 7370 6574 7461 2069
6c20 7075 6c73 616e 7465 2073 696e 6973
7472 6f20 6465 6c20 6d6f 7573 650a 0962
6e65 2e73 0957 4c4d 420a 0a09 6d6f 7665
2e77 0923 2430 3031 302c 2439 3628 6136
2909 093b 7370 6567 6e65 2055 5345 3050
310a 096f 722e 7709 2324 3830 3030 2c64
3609 093b 6163 6365 6e64 6520 696c 2062
6974 2031 3520 2853 4554 2f43 4c52 290a
096d 6f76 652e 7709 6436 2c24 3965 2861
3629 0909 3b72 6569 6d70 6f73 7461 2041
444b 434f 4e20 6465 6c6c 274f 530a 096d
6f76 652e 7709 2324 3030 3033 2c24 3936
2861 3629 0909 3b73 7065 676e 6520 4155
4430 2065 2041 5544 310a 096f 722e 7709
2324 3830 3030 2c64 3709 093b 6163 6365
6e64 6520 696c 2062 6974 2031 3520 2853
4554 2f43 4c52 290a 096d 6f76 652e 7709
6437 2c24 3936 2861 3629 0909 3b72 6569
6d70 6f73 7461 2044 4d41 2064 656c 6c27
4f53 0a09 6d6f 7665 2e6c 0934 2e77 2c61
360a 096a 7372 095f 4c56 4f45 6e61 626c
6528 6136 290a 0972 7473 0a0a 0953 4543
5449 4f4e 0953 616d 706c 652c 4441 5441
5f43 093b 7665 6e65 6e64 6f20 6c65 7474
6120 6461 6c20 444d 4120 6465 7665 2065
7373 6572 6520 696e 2043 4849 500a 0a41
726d 6f6e 6963 613a 093b 6172 6d6f 6e69
6361 2064 6920 3136 2076 616c 6f72 6920
6372 6561 7461 2063 6f6c 2749 4320 6465
6c20 7472 6173 6827 6d2d 6f6e 650a 0944
432e 4209 2431 392c 2434 362c 2436 392c
2437 432c 2437 442c 2436 412c 2434 372c
2431 412c 2445 382c 2442 422c 2439 372c
2438 342c 2438 332c 2439 352c 2442 382c
2445 350a 4d6f 6446 713a 0a09 626c 6b2e
7709 3530 302a 320a 4d6f 6446 715f 656e
643a 0a09 454e 440a 0a51 7565 7374 6120
766f 6c74 6120 6c27 4155 4431 564f 4c20
e820 7374 6174 6f20 696d 706f 7374 6174
6f2c 2063 6f73 6120 6368 6520 6e6f 6e20
e820 7375 6363 6573 7361 2070 6572 f220
7065 720a 6c27 4155 4431 5045 5220 6368
6520 7669 656e 6520 636f 6e74 696e 7561
6d65 6e74 6520 6d6f 6469 6669 6361 746f
2064 616c 2063 616e 616c 6520 6d6f 6475
6c61 746f 7265 3b20 6c27 4155 4430 5045
520a e820 696e 7665 6365 2073 7461 746f
2069 6d70 6f73 7461 746f 2061 6666 696e
6368 e820 6c65 6767 6573 7365 206c 6120
7461 6265 6c6c 6120 6469 2031 3030 3020
776f 7264 2069 6e20 3220 7365 636f 6e64
690a 636f 6e20 6c61 2063 6f73 7461 6e74
6520 6469 2063 6c6f 636b 2064 6976 6973
6120 7065 7220 6d65 74e0 2064 656c 6c61
206c 756e 6768 657a 7a61 2064 656c 6c61
```

```
2074 6162 656c 6c61 2c0a 6c65 6767 656e
646f 2071 7569 6e64 6920 6164 2031 2048
7a20 3530 3020 776f 7264 2c20 6f76 7665
726f 206c 6567 6765 6e64 6f6c 6120 7475
7474 6120 696e 2022 6d65 7a7a 6f22 2048
7a2e 0a
```

Lezione14-7c

```
f9fa f9fa 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0a3b 2043
6f72 736f 2041 736d 202d 204c 455a 494f
4e45 2078 783a 2020 2a20 4d4f 4455 4c41
5245 2049 4e20 414d 5049 455a 5a41 2045
4420 494e 2046 5245 5155 454e 5a41 2055
4e27 4152 4d4f 4e49 4341 202a 0a0a 0953
4543 5449 4f4e 094c 455a 494f 4e45 7878
3762 2c43 4f44 450a 0a53 7461 7274 3a0a
0a09 6c65 6109 6d6f 6476 6f6c 6671 2c61
300a 096d 6f76 6571 0923 302c 6430 0a09
6d6f 7665 7109 2331 3233 2c64 310a 096d
6f76 652e 7709 2336 342d 312c 6437 0a2e
4c70 313a 096d 6f76 652e 7709 6430 2c28
6130 292b 0a09 6164 6471 2e77 0923 312c
6430 0a09 6d6f 7665 2e77 0964 312c 2861
3029 2b0a 0961 6464 712e 7709 2334 2c64
310a 0964 6272 6109 6437 2c2e 6c70 310a
096d 6f76 652e 7709 2336 342d 312c 6437
0a2e 4c70 323a 096d 6f76 652e 7709 6430
2c28 6130 292b 0a09 7375 6271 2e77 0923
312c 6430 0a09 6d6f 7665 2e77 0964 312c
2861 3029 2b0a 0973 7562 712e 7709 2334
2c64 310a 0964 6272 6109 6437 2c2e 6c70
320a 0a5f 4c56 4f44 6973 6162 6c65 0945
5155 092d 3132 300a 5f4c 564f 456e 6162
6c65 0945 5155 092d 3132 360a 0a09 6d6f
7665 2e6c 0934 2e77 2c61 360a 096a 7372
095f 4c56 4f44 6973 6162 6c65 2861 3629
0a0a 0962 7365 7409 2331 2c24 6266 6530
3031 0909 3b73 7065 676e 6520 696c 2066
696c 7472 6f20 7061 7373 612d 6261 7373
6f0a 0a09 6c65 6109 2464 6666 3030 302c
6136 0a09 6d6f 7665 2e77 0924 3228 6136
292c 6437 0909 3b73 616c 7661 2044 4d41
2064 656c 6c27 4f53 0a09 6d6f 7665 2e77
0924 3130 2861 3629 2c64 3609 093b 7361
6c76 6120 4144 4b43 4f4e 2064 656c 6c27
4f53 0a0a 436c 6f63 6b09 6571 7509 3335
3436 3839 350a 0a09 6d6f 7665 2e6c 0923
6172 6d6f 6e69 6361 2c24 6230 2861 3629
0a09 6d6f 7665 2e77 0923 3136 2f32 2c24
6234 2861 3629 0a0a 096d 6f76 652e 6c09
236d 6f64 766f 6c66 712c 2461 3028 6136
290a 096d 6f76 652e 7709 2328 6d6f 6476
6f6c 6671 5f65 6e64 2d6d 6f64 766f 6c66
7129 2f32 2c24 6134 2861 3629 0a09 6d6f
7665 2e77 0923 636c 6f63 6b2f 2828 6d6f
6476 6f6c 6671 5f65 6e64 2d6d 6f64 766f
6c66 7129 2f32 292c 2461 3628 6136 290a
0a09 6d6f 7665 2e77 0923 2438 3031 312c
```

```
2439 6528 6136 2909 093b 696d 706f 7374
6120 5553 4530 5631 2065 2055 5345 3050
310a 0a09 6d6f 7665 2e77 0923 2438 3230
332c 2439 3628 6136 2909 093b 6163 6365
6e64 6520 4155 4430 2065 2041 5544 3120
696e 2044 4d41 434f 4e57 0a0a 574c 4d42
3a09 6274 7374 0923 362c 2462 6665 3030
3109 093b 6173 7065 7474 6120 696c 2070
756c 7361 6e74 6520 7369 6e69 7374 726f
2064 656c 206d 6f75 7365 0a09 626e 652e
7309 574c 4d42 0a0a 096d 6f76 652e 7709
2324 3030 3131 2c24 3936 2861 3629 0909
3b73 7065 676e 6520 5553 4530 5631 2065
2055 5345 3050 310a 096f 722e 7709 2324
3830 3030 2c64 3609 093b 6163 6365 6e64
6520 696c 2062 6974 2031 3520 2853 4554
2f43 4c52 290a 096d 6f76 652e 7709 6436
2c24 3965 2861 3629 0909 3b72 6569 6d70
6f73 7461 2041 444b 434f 4e20 6465 6c6c
274f 530a 096d 6f76 652e 7709 2324 3030
3033 2c24 3936 2861 3629 0909 3b73 7065
676e 6520 4155 4430 2065 2041 5544 310a
096f 722e 7709 2324 3830 3030 2c64 3709
093b 6163 6365 6e64 6520 696c 2062 6974
2031 3520 2853 4554 2f43 4c52 290a 096d
6f76 652e 7709 6437 2c24 3936 2861 3629
0909 3b72 6569 6d70 6f73 7461 2044 4d41
2064 656c 6c27 4f53 0a09 6d6f 7665 2e6c
0934 2e77 2c61 360a 096a 7372 095f 4c56
4f45 6e61 626c 6528 6136 290a 0972 7473
0a0a 0953 4543 5449 4f4e 0953 616d 706c
652c 4441 5441 5f43 093b 7665 6e65 6e64
6f20 6c65 7474 6120 6461 6c20 444d 4120
6465 7665 2065 7373 6572 6520 696e 2043
4849 500a 0a41 726d 6f6e 6963 613a 093b
6172 6d6f 6e69 6361 2064 6920 3136 2076
616c 6f72 6920 6372 6561 7461 2063 6f6c
2749 4320 6465 6c20 7472 6173 6827 6d2d
6f6e 650a 0944 432e 4209 2431 392c 2434
362c 2436 392c 2437 432c 2437 442c 2436
412c 2434 372c 2431 412c 2445 382c 2442
422c 2439 372c 2438 342c 2438 332c 2439
352c 2442 382c 2445 350a 4d6f 6456 6f6c
4671 3a0a 0962 6c6b 2e77 0936 342a 322a
320a 4d6f 6456 6f6c 4671 5f65 6e64 3a0a
0945 4e44 0a0a 4563 636f 2063 6f6d 6520
6d6f 6475 6c61 7265 2073 6961 2069 6e20
616d 7069 657a 7a61 2063 6865 2069 6e20
6672 6571 7565 6e7a 6120 756e 2073 756f
6e6f 3a20 6c61 2074 6162 656c 6c61 0a63
6f6e 7369 7374 6520 6469 2032 2076 616c
6f72 6920 616c 7465 726e 6174 692c 2070
7269 6d61 2075 6e61 2077 6f72 6420 636f
6e20 696c 2076 6f6c 756d 6520 6120 3720
6269 7420 7065 720a 4155 4431 564f 4c20
6520 706f 6920 756e 6120 7365 636f 6e64
6120 776f 7264 2063 6f6e 2069 6c20 7065
7269 6f64 6f20 6120 3136 2062 6974 2070
6572 2041 5544 3050 4552 3b20 636f 6e20
6c61 0a6d 6573 6564 696d 6120 7375 6363
6573 696f 6e65 2069 2064 6174 6920 656e
7472 616e 6f20 616e 6368 6520 696e 2041
```

```
5544 3044 4154 3a20 7072 696d 6120 696c
2076 6f6c 756d 652c 2070 6f69 0a69 6c20
7065 7269 6f64 6f2e 0a4e 6174 7572 616c
656d 656e 7465 2c20 736f 6e6f 2073 7461
7469 2069 6d70 6f73 7461 7469 2073 6961
2069 6c20 6269 7420 6469 206d 6f64 756c
617a 696f 6e65 2064 6920 6672 6571 7565
6e7a 6120 6368 650a 6469 2061 6d70 6965
7a7a 6120 6465 6c20 6361 6e61 6c65 2030
206e 6569 2063 6f6e 6672 6f6e 7469 2064
656c 2063 616e 616c 6520 312e 0a
```

28.8 Lezione14-8

```
; Corso Asm - LEZIONE xx:  * MIXARE 2 SAMPLE *

        section bau,code

Start:

_LVODisable     EQU     -120
_LVOEnable      EQU     -126

        move.l  4.w,a6
        jsr     _LVODisable(a6)
        bset    #1,$bfe001              ;spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ;salva DMA dell'OS

        move.l  #sample1,$a0(a6)
        move.l  #sample2,$b0(a6)
        move.w  #(sample1_end-sample1)/2,$a4(a6)
        move.w  #(sample2_end-sample2)/2,$b4(a6)
Clock   equ     3546895
        move.w  #clock/21056,$a6(a6)
        move.w  #clock/21056,$b6(a6)
        move.w  #64,$a8(a6)
        move.w  #64,$b8(a6)
        move.w  #$8003,$96(a6)          ;accende AUD0-AUD1 DMA in DMACONW

WLMB:   btst    #6,$bfe001
        bne.s   wlmb

        lea     sample0,a0
        move.l  #sample0_end-sample0,d0
        lea     sample1,a1
        move.l  #sample1_end-sample1,d1
        lea     sample2,a2
        move.l  #sample2_end-sample2,d2
        bsr.s   mixsamples
        move.l  #sample0,$a0(a6)        ;verrà playato alla fine dei sample 1
        move.l  #sample0,$b0(a6)        ;e 2: vi ricordate perchè ?
        move.w  #(sample0_end-sample0)/2,$a4(a6)
        move.w  #(sample0_end-sample0)/2,$b4(a6)

WRMB:   btst    #10,$dff016
        bne.s   wrmb

        move.w  #$0003,$96(a6)          ;spegne i DMA
        or.w    #$8000,d7               ;accende il bit 15 (SET/CLR)
```

```
        move.w  d7,$96(a6)              ;reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     _LVOEnable(a6)
        rts


MixSamples:     ;[a0=dst sample, a1=src sample 1, a2=soruce sample 2]
                ;[d0.l=dst length.b, d1.l=src1 length.b, d2.l=src2 length.b]
        movem.l d0-d3/a0-a4,-(sp)
        lea     (a1,d1.l),a3            ;a3=fine del sample 1
        lea     (a2,d2.l),a4            ;a4=fine de sample 2
        moveq   #0,d3                   ;d3.b=0 per ADDX
.Lp:    move.w  #$f00,$dff180
        move.b  (a1)+,d1                ;d1.b=campione del sample 1
        ext.w   d1                      ;d1.w=d1.b esteso di segno a word
        move.b  (a2)+,d2                ;d2.b=campione del sample 2
        ext.w   d2                      ;d2.w=d2.b esteso si segno a word
        add.w   d1,d2                   ;d2.w=somma CON SEGNO i campioni 1 e 2
        asr.w   #1,d2                   ;d2.w=media dei campioni (somma/2)
        addx.b  d3,d2                   ;d2.w=campione mixato ARROTONDATO per
                                        ;eccesso o per difetto in base al bit
                                        ;uscito con l'ASR
        move.b  d2,(a0)+                ;salva il campione mixato
        cmp.l   a3,a1                   ;è finito il sample 1 ?
        bhs.s   .quit                   ;se SI: esci
        cmp.l   a4,a2                   ;è finito il sample 2 ?
        bhs.s   .quit                   ;se SI: esci
        subq.l  #1,d0                   ;decrementa lungh0.b fino a 0...senza
                                        ;DBRA perchè funziona solo a word...
        bhi.s   .lp
.Quit:  movem.l (sp)+,d0-d3/a0-a4
        rts


        SECTION Sample,DATA_C

Sample1:
        incbin  "assembler2:sorgenti8/carrasco.21056"
Sample1_end:

Sample2:
        incbin  "assembler2:sorgenti8/lee3.21056"
Sample2_end:

Sample0:blk.b   sample1_end-sample1
Sample0_end:

        END
```

Cosa abbiamo combinato questa volta ? Siamo riusciti a suonare 2 sample diversi sulla stessa voce ! Come ? Mixandoli via software con la CPU !
Conoscete bene la struttura della forma d'onda id una sample, e sapete che ogni campione di 1 byte può variare da -128 a 127, pertanto sono BYTE CON SEGNO con i quali è possibile lavorare trattandoli secndo la loro natura di numeri ad 8 bit, il più significativo dei quali agisce da segno.
Quale miglior metodo per fare in modo che date due serie di numeri se ne ottenga una che ricalchi l'andamento di entrambe ?
Fare la MEDIA ARITMETICA tra ogni coppia di singoli byte/campioni: prendendo 2 campioni corrispondenti dell'uno e dell'altro sample, è sufficiente sommarli algebricamente (* TENENDO QUINDI CONTO DEL SEGNO *) e dividere il risultato per 2: MIX = (SAMP1 + SAMP2) / 2.

```
Quando si sommano algebricamente due byte, è possibile che il risultato sia
maggiore di 127 (ad esempio, se entrambi sono 127, la somma sarà 254), e,
pertanto, non esprimibile con un numero ad 8 bit con segno, per cui è
necessario lavorare a word, per calcolare la media, e le word devono anch'esse
rispecchiare il segno dei byte originali: per questo motivo abbiamo esteso
il segno del byte alla word per fare la somma algebrica con ADD.W.
Nella ruotine "MixSamples" abbiamo adottato un ulteriore particolare per
incrementare la qualità e la precisione di mixing: l'ARROTONDAMENTO.
Una volta fatta la somma, è necessario dividere per 2, affinchè i valori
ritornino nel range di 8 bit con segno (* bisogna dividere tutti i valori per
2, non omettere quelli che sono compresi tra -128 e 127 anche solo dopo aver
fatto l'ADD: i campioni NON SAREBBERO PIU' PROPORZIONALI ! *); tale divisione
viene eseguita velocemente dall'ASR, che shifta a destra (di 1, in questo caso)
* MANTENENDO il segno a sinistra *: l'ultimo bit che esce da destra dal
registro shiftato è contenuto nel flag X (eXtend) della CPU; ora quel bit
è come una sorta di "valore oltre la virgola" che sprime l'approssimazione
dell'"intero" contenuto nel registro: aggiungento il contenuto del flag X
all'intero si arrotondano tutti i numeri originariamente dispari al numero
succesivo. Per sempio: 17 + 6 = 23, 23 / 2 = 11.5 (=%x.1) = 12 (arrotondato);
od ancora: 11 + 23 = 34, 34 / 2 = 17 (%x.0) = 17 (arrotondato).

N.B.:   avete notato che il volume (inteso come livello medio dei campioni)
        del sample mixato è inferiore alla resa dei 2 sample letti
        contemporaneamente ? Come mai ? La risposta alla prossima puntata...
```

Lezione14-8b

```
; Corso Asm - LEZIONE xx:  * MIXARE 2 SAMPLE E BOOSTARE IL VOLUME *

Start:

_LVODisable     EQU     -120
_LVOEnable      EQU     -126

        move.l  4.w,a6
        jsr     _LVODisable(a6)
        bset    #1,$bfe001              ;spegne il filtro passa-basso
        lea     $dff000,a6
        move.w  $2(a6),d7               ;salva DMA dell'OS


        move.l  #sample1,$a0(a6)
        move.l  #sample2,$b0(a6)
        move.w  #(sample1_end-sample1)/2,$a4(a6)
        move.w  #(sample2_end-sample2)/2,$b4(a6)
Clock   equ     3546895
        move.w  #clock/21056,$a6(a6)
        move.w  #clock/21056,$b6(a6)
        move.w  #64,$a8(a6)
        move.w  #64,$b8(a6)
        move.w  #$8003,$96(a6)          ;accende AUD0-AUD1 DMA in DMACONW

WLMB:   btst    #6,$bfe001
        bne.s   wlmb

        lea     sample0,a0
        move.l  #sample0_end-sample0,d0
        lea     sample1,a1
        move.l  #sample1_end-sample1,d1
```

```
        lea     sample2,a2
        move.l  #sample2_end-sample2,d2
        bsr.s   boost_mixsamples

        lea     $dff000,a6
        move.l  #sample0,$a0(a6)
        move.l  #sample0,$b0(a6)
        move.w  #(sample0_end-sample0)/2,$a4(a6)
        move.w  #(sample0_end-sample0)/2,$b4(a6)
        move.w  #$8003,$96(a6)

WRMB:   btst    #10,$dff016
        bne.s   wrmb

        move.w  #$0003,$96(a6)          ;spegne i DMA
        or.w    #$8000,d7               ;accende il bit 15 (SET/CLR)
        move.w  d7,$96(a6)              ;reimposta DMA dell'OS
        move.l  4.w,a6
        jsr     _LVOEnable(a6)
        rts


Boost_MixSamples:
                ;[a0=dst sample, a1=src sample 1, a2=soruce sample 2]
                ;[d0.l=dst length.b, d1.l=src1 length.b, d2.l=src2 length.b]
        movem.l d0-d3/a0-a4,-(sp)
        lea     (a1,d1.l),a3            ;a3=fine del sample 1
        lea     (a2,d2.l),a4            ;a4=fine de sample 2
        moveq   #0,d3                   ;d3.w=0=MAX campione di partenza
.Lp1:   move.w  #$f00,$dff180
        move.b  (a1)+,d1                ;d1.b=campione del sample 1
        ext.w   d1                      ;d1.w=d1.b esteso di segno a word
        move.b  (a2)+,d2                ;d2.b=campione del sample 2
        ext.w   d2                      ;d2.w=d2.b esteso si segno a word
        add.w   d1,d2                   ;d2.w=somma CON SEGNO i campioni 1 e 2
        bpl.s   .noabs
        neg.w   d2
.NoAbs: cmp.w   d3,d2                   ;d2.w=valore assoluto di d2
        bls.s   .nomax
        move.w  d2,d3                   ;se d2>d3: d3(MAX)=d2
.NoMax: cmp.l   a3,a1                   ;è finito il sample 1 ?
        bhs.s   .quit1                  ;se SI: esci
        cmp.l   a4,a2                   ;è finito il sample 2 ?
        bhs.s   .quit1                  ;se SI: esci
        subq.l  #1,d0
        bhi.s   .lp1
.Quit1: move.l  (sp),d0                 ;ripristina d0
        movem.l 5*4(sp),a1-a2           ;ripristina a1 ed a2
        move.w  d3,$7ff0000
                                        ;d3.w=MAX raggiunto dalle somme
.Lp2:   move.w  #$00f,$dff180
        move.b  (a1)+,d1                ;d1.b=campione del sample 1
        ext.w   d1                      ;d1.w=d1.b esteso di segno a word
        move.b  (a2)+,d2                ;d2.b=campione del sample 2
        ext.w   d2                      ;d2.w=d2.b esteso si segno a word
        add.w   d1,d2                   ;d2.w=somma CON SEGNO i campioni 1 e 2

        muls.w  #127,d2                 ;PROPORZIONE: d3(MAX)/127=d2/x
        divs.w  d3,d2
        move.b  d2,(a0)+

        cmp.l   a3,a1                   ;è finito il sample 1 ?
```

```
        bhs.s   .quit2                  ;se SI: esci
        cmp.l   a4,a2                   ;è finito il sample 2 ?
        bhs.s   .quit2                  ;se SI: esci
        subq.l  #1,d0                   ;decrementa lungh0.b fino a 0...senza
        bhi.s   .lp2
.Quit2: movem.l (sp)+,d0-d3/a0-a4
        rts


        SECTION Sample,DATA_C

Sample1:
        incbin  "assembler2:sorgenti8/carrasco.21056"
Sample1_end:

Sample2:
        incbin  "assembler2:sorgenti8/lee3.21056"
Sample2_end:

Sample0:blk.b   sample1_end-sample1
Sample0_end:
        END
```

In teoria, il vero mixing si dovrebbe fare solo sommando algebricamente i campioni, tuttavia, per ovvi omtivi, spesso si fuoriesce dal range di 8 bit con segno, e, per deamplificare direttamente la forma d'onda in maniera equa, si divide sempre per 2. Risultato: la resa d'intensità finale è minore di quella dei 2 sample suonati indipendentemente su due canali diversi.
Per poter utilizzare il normale algoritmo di mixing, sarebbe necessario che la somma non sorpassi mai 127 o -128, che non esca mai dai limiti del range. Visto che non conviene campionare sample bassi perchè l'audio ad 8 bit non possiede una grande precisione, siamo costretti a boostare il volume del sample mixato fino al limite: si considera il più alto volume raggiunto dalle somme, e lo si usa come range massimo proporzionale a 127 (valore assoluto massimo raggiungibile: NON 128, poichè solo la parte negativa arriva a -128, ed amplificando troppo la positiva - oltre 127 - saremmo al punto di prima). Le proporzioni - che, personalmente, sono solito definire « gli "zoom" della matematica » - sono utili, in questo caso, per restringere il campo/range entro i limiti in modo appunto proporzionale ed equo per tutti i sample.
*** In sostanza, abbiamo deamplificato fino a che il valore più alto delle somme non fosse pari a 127 (o -127), e tutto il resto proporzionalmente ***.

N.B.: anche se sarebbe forse stato opportuno, non è stato applicato alcun arrotondamento: si sarebbe dovuto usare un sorta di approssimazione oltre la virgola di più bit, che, pur incrementando - anche se non effettivamente percepibilmente - la qualità del mixing, avrebbe causato non pochi problemi di comprensione del sorgente e, soprattutto, di velocità: dopo la moltiplicazione per 127 avremmo potuto shiftare il tutto a destra di 16 bit (moltiplicando il numero di molto per simulare la virgola con numeri molto grandi da dividere, poi), ottenendo così un valore a 32 bit; valore che doveva essere diviso per MAX e poi risfhiftato a sinistra di 16 bit ed arrotondato con il bit più significativo della parte shiftata per essere riportato alla grandezza originale. Tutto ciò, però, comporterebbe un problema dovuto ad una limitazione del 68000: dividendo il valore a 32 bit per MAX, il risultato - se MAX è vicino al valore somma corrente - potrebbe essere ancora a 32, e l'istruzione DIVS - purtroppo - restituisce il risultato a 16 bit nella parte bassa del registro ed il resto nella parte alta, cancellando la word a noi tanto utile. Il problema si sarebbe potuto presentare per qualsiasi altra approssimazione...

28.9 Lezione14-9b

```
; Corso Asm:    LEZIONE14 - ** (IN)UTILITY DI DIAL DA CLI **

; Questo sorgente va assemblato e reso eseguibile, poichè funziona come
; comando da CLI: es) > Dial 113[RETURN]
; Assemblare con 'a' e salvare con 'wo'

        SECTION LEZIONE14-9b,CODE

main:
        moveq   #0,d0
        move.b  (a0)+,d0                ;l'OS punta a0 alla stringa ASCII
                                        ;dei parametri dopo il comando CLI
        cmp.b   #10,d0                  ;loopa fino al return (ASCII=10)
        beq.b   .esci
        sub.w   #'0',d0
        add.w   d0,d0
        add.w   d0,d0
        move.l  Numbers(pc,d0.w),d0
        bsr.s   Start
        bra.s   main
.esci:  moveq   #0,d0
        rts

        even
Numbers:

zero:           dc.l    $ee00a7         ; 0
uno:            dc.l    $14100b9        ; 1
due:            dc.l    $14100a7        ; 2
tre:            dc.l    $1410097        ; 3
quattro:        dc.l    $12200b9        ; 4
cinque:         dc.l    $12200a7        ; 5
sei:            dc.l    $1220097        ; 6
sette:          dc.l    $10700b9        ; 7
otto:           dc.l    $10700a7        ; 8
nove:           dc.l    $1070097        ; 9
cancelletto:    dc.l    $ee0097         ; #
asterisco:      dc.l    $ee00b7         ; *

; d0.l =        numero da suonare

_LVODisable     EQU     -120
_LVOEnable      EQU     -126

Start:
        move.l  4.w,a6
        jsr     _LVODisable(a6)

        lea     $dff000,a5      ; Custom base per offsets

        move.l  #SMP1,$a0(a5)           ; AUD0LCH - indirizzo del sample
        move.l  #SMP1,$b0(a5)           ; AUD1LCH - indirizzo del sample
        move.l  #SMP1,$c0(a5)           ; AUD2LCH - indirizzo del sample
        move.l  #SMP1,$d0(a5)           ; AUD3LCH - indirizzo del sample
        move.w  #SMP1LEN,$A4(a5)        ; AUD0LEN - lunghezza del sample
        move.w  #SMP1LEN,$B4(a5)        ; AUD1LEN - lunghezza del sample
        move.w  #SMP1LEN,$C4(a5)        ; AUD2LEN - lunghezza del sample
        move.w  #SMP1LEN,$D4(a5)        ; AUD3LEN - lunghezza del sample
        move.w  #64,$A8(a5)             ; AUD0VOL - volume massimo
```

```
        move.w  #64,$B8(a5)             ; AUD1VOL - volume massimo
        move.w  #64,$C8(a5)             ; AUD2VOL - volume massimo
        move.w  #64,$D8(a5)             ; AUD3VOL - volume massimo
        move.w  d0,$c6(a5)              ; AUD2PER - period
        move.w  d0,$d6(a5)              ; AUD3PER - period
        swap    d0
        move.w  d0,$a6(a5)              ; AUD0PER - period
        move.w  d0,$b6(a5)              ; AUD1PER - period

        move.w  $2(a5),d7
        move.w  #$820f,$96(a5)  ; MACON - abilito il canale DMA audio 0,
                                ; dunque il sample comincia ad essere suonato.
        MOVEQ   #12,D1          ; numero di fotogrammi da aspettare
        MOVEQ   #-1,D0
WBLAN1: CMP.B   6(A5),d0
        BNE.S   WBLAN1
WBLAN2: CMP.B   6(A5),D0
        BEQ.S   WBLAN2
        DBRA    D1,WBLAN1

        or.w    #$8000,d7
        move.w  #$000f,$96(a5)
        move.w  d7,$96(a5)      ; DMACON - chiudo il canale audio 0 (zitto!)

        MOVEQ   #4,D1           ; numero di fotogrammi da aspettare
        MOVEQ   #-1,D0
WBLAN1b:CMP.B   6(A5),d0
        BNE.S   WBLAN1b
WBLAN2b:CMP.B   6(A5),D0
        BEQ.S   WBLAN2b
        DBRA    D1,WBLAN1b

        jsr     _LVOEnable(a6)
        rts

        SECTION Tono,DATA_C

SMP1:
        dc.b $00,$31,$5a,$75,$7f,$75,$5a,$31,$00,$cf,$a6,$8b,$81,$8b,$a6,$cf
SMP1LEN equ     (*-SMP1)>>1     ; lunghezza del sample in words

        END
```

Adesso abbiamo toccato il fondo: ecco a voi una perfetta INUTILITY !!!
Un programmino eseguibile da CLI che serve a comporre numeri telefonici
a toni: non serve a nulla, ma, se non altro, avete imparato come il sistema
operativo passa la linea dei paramentri di un comando CLI...

28.10 Lezione14-10a

```
;  Lezione14-10a.s - Uso della routine player6.1a per un modulo non compresso

; Routine P61_Music chiamata ogni vertical blank

        SECTION Usoplay61a,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
```

```
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;
; Call P61_Init to initialize the playroutine
; D0 --> Timer detection (for CIA-version)
; A0 --> Address to the module
; A1 --> Address to samples/0
; A2 --> Address to sample buffer
; D0 <-- 0 if succeeded
; A6 <-- $DFF000
;

        movem.l d0-d7/a0-a6,-(SP)
        lea     P61_data,a0     ; Indirizzo del modulo in a0
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
        sub.l   a2,a2           ; no samples -> modulo non compattato
        bsr.w   P61_Init
        movem.l (SP)+,d0-d7/a0-a6

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.w  #$e000,$9a(a5)          ; INTENA - Abilito Master and lev6
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$08000,d2      ; linea da aspettare = $80
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

;
; Call P61_Music every frame to play the music
;        _NOT_ if CIA-version is used!
; A6 --> Customchip baseaddress ($DFF000)
;

        move.w  #$f00,$180(a5)  ; color0 rosso -> per fare il copper monitor

        movem.l d0-d7/a0-a6,-(SP)
```

```
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_Music
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$003,$180(a5)  ; color0 nero

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

;
; Call P61_End to stop the music
;   A6 --> Customchip baseaddress ($DFF000)
;               Uses D0/D1/A0/A1/A3
;

        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_End

        rts


*****************************************************************************
*                The Player 6.1A for Asm-One 1.09 and later                 *
*****************************************************************************

fade  = 0       ;0 = Normal, NO master volume control possible
                ;1 = Use master volume (P61_Master)

jump = 0        ;0 = do NOT include position jump code (P61_SetPosition)
                ;1 = Include

system = 0      ;0 = killer
                ;1 = friendly

CIA = 0         ;0 = CIA disabled
                ;1 = CIA enabled

exec = 1        ;0 = ExecBase destroyed
                ;1 = ExecBase valid

opt020 = 0      ;0 = MC680x0 code
                ;1 = MC68020+ or better

use = $2009559  ; Usecode (mettete il valore dato dal p61con al salvataggio
                ; diverso per ogni modulo!)

*****************************************************************************
        include "play.s"        ; La routine vera e propria!
*****************************************************************************


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $180,$003       ; color0 nero
        dc.W    $FFFF,$FFFE     ; fine della copperlist
```

```
*****************************************************************************
;       Modulo musicale convertito in formato P61, non compresso
*****************************************************************************

        Section modulozzo,data_C

; Il modulo e' di DreamFish. Originale 42684, convertito 31628 (non packed!)

P61_data:
        incbin  "P61.technochild"       ; non compresso, solo convertito.

        end
```

Vi ricordo i passaggi per suonare un modulo con questa replayroutine: come
prima cosa dovete convertire il modulo in formato P61 con l'apposita utility
P61CON, lasciando nelle preferences del programma le varie opzioni "delta",
"pack samples", ... tutte azzerate, eccetto "tempo".
In questo modo otterrete il modulo convertito adatto per essere risuonato
da questa routine. Normalmente nella conversione si risparmia anche spazio,
ma non si tratta di una "compressione", bensi' di una ottimizzazione.
Annotatevi l'usecode, perche' lo dovete mettere nel listato all'equate "use".
Questo serve per risparmiare spazio: infatti indica quali effetti sono usati
dal modulo, in modo da non assemblare quelli non usati.
A questo punto, basta includere in chip ram il modulo, e chiamare le
routines al momento giusto: P61_Init prima di suonare, come facevamo con
mt_init, P61_Music una volta ogni vertical blank (aspettando con $dff004/6 o
mettendolo in interrupt $6c), e prima di uscire P61_End.
NON DIMENTICATEVI DI ABILITARE L'INTERRUPT DI LIVELLO 6!!!! E' CON QUELLO CHE
VENGONO FATTE ALCUNE TEMPORIZZAZIONI, usando l'interrupt del timer A del CIAB.
Quindi, non potete usare il timer A del CIAB mentre si usa questa routine...
A dire il vero, anche usare il timer B del CIAB potrebbe portare problemi...
Quindi, attenti! Settate i bit 15,14,13 del $dff09a (intena), oltre agli
eventuali bit 5 o 4 (VERTB e COPER), e non usate il timer A/CIAB.
Naturalmente ci sono altri particolari, come il ricordarsi di mettere i
valori giusti nei registri ($dff000 in a6, eccetera), e settare gli equate
"fade", "jump", "system", "cia", "exec", "opt020", "use" nel modo giusto.
A questo proposito, ci sono vari esempi con gli equate settati a secondo
delle varie esigenze. In particolare, in questo esempio abbiamo CIA = 0,
perche' chiamiamo ogni volta P61_music, come sempre.

Lezione14-10b

```
;  Lezione14-10b.s - Uso della routine player6.1a per un modulo compresso

; Routine P61_Music chiamata ogni vertical blank

        SECTION Usoplay61a,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
```

```
;
; Call P61_Init to initialize the playroutine
; D0 --> Timer detection (for CIA-version)
; A0 --> Address to the module
; A1 --> Address to samples/0
; A2 --> Address to sample buffer
; D0 <-- 0 if succeeded
; A6 <-- $DFF000
;

        movem.l d0-d7/a0-a6,-(SP)
        lea     P61_data,a0     ; Indirizzo del modulo in a0
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
        lea     samples,a2      ; modulo compattato! Buffer destinazione per
                                ; i samples (in chip ram) da indicare!
        bsr.w   P61_Init        ; Nota: impiega alcuni secondi per decompress!
        movem.l (SP)+,d0-d7/a0-a6

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.w  #$e000,$9a(a5)          ; INTENA - Abilito Master and lev6
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$08000,d2      ; linea da aspettare = $80
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

;
; Call P61_Music every frame to play the music
;        _NOT_ if CIA-version is used!
; A6 --> Customchip baseaddress ($DFF000)
;
        move.w  #$f00,$180(a5)  ; color0 rosso -> per fare il copper monitor

        movem.l d0-d7/a0-a6,-(SP)
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_Music
        movem.l (SP)+,d0-d7/a0-a6

        move.w  #$003,$180(a5)  ; color0 nero

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse
```

```
;
; Call P61_End to stop the music
;   A6 --> Customchip baseaddress ($DFF000)
;               Uses D0/D1/A0/A1/A3
;

        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_End

        rts


*****************************************************************************
*               The Player 6.1A for Asm-One 1.09 and later                  *
*****************************************************************************

fade  = 0       ;0 = Normal, NO master volume control possible
                ;1 = Use master volume (P61_Master)

jump = 0        ;0 = do NOT include position jump code (P61_SetPosition)
                ;1 = Include

system = 0      ;0 = killer
                ;1 = friendly

CIA = 0         ;0 = CIA disabled
                ;1 = CIA enabled

exec = 1        ;0 = ExecBase destroyed
                ;1 = ExecBase valid

opt020 = 0      ;0 = MC680x0 code
                ;1 = MC68020+ or better

use = $b55a     ; Usecode (mettete il valore dato dal p61con al salvataggio
                ; diverso per ogni modulo!)

*****************************************************************************
        include "play.s"        ; La routine vera e propria!
*****************************************************************************


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $180,$003       ; color0 nero
        dc.W    $FFFF,$FFFE     ; fine della copperlist

*****************************************************************************
;       Modulo musicale convertito in formato P61, COMPRESSO! (opzione pack!)
*****************************************************************************

        Section modulozzo,data  ; Non occorre sia in chip ram, perche' e'
                                ; compresso e sara' scompattato altrove!

; Il modulo e' di Jester/Sanity. Originale 153676, packed 71950
```

```
P61_data:
        incbin  "P61.stardust"  ; Compresso, (opzione PACK SAMPLES), per cui
                                ; si puo' mettere anche in fast ram: sara'
                                ; usato per scompattare i samples nel buffer
                                ; samples, e non sara' "suonato" direttamente,
                                ; quindi non dovra' passare per i canali DMA
                                ; audio, ma solo dalla routine di depack del
                                ; processore. Quindi, basta un DATA (non _C!)


*****************************************************************************
;       Dove saranno scompattati i samples (section bss in chip ram!)
*****************************************************************************

        section smp,bss_c

samples:
        ds.b    132112  ; lunghezza riportata dal p61con

        end
```

Questo esempio e' come quello precedente, solo che il modulo ha i sample compressi (opzione "Pack Samples" attiva, ma senza "Delta" attivo, che pero' che sia attivo o no ho notato che non cambia quasi niente!!!)
Prima di prendere la decisione di compattare il sample, pensateci due o tre volte, infatti occorre usare piu' memoria, per creare un buffer dove mettere i sample scompattati, si perde del tempo nello scompattarli, e si puo' perdere qualita' audio. A tal proposito, se si sceglie di compattare i sample di un modulo, appare un requester che ci permette di scegliere sample per sample quali compattare e quali no, e di ascoltare l'originale e l'eventuale versione compattata. Ascoltando i vari sample in versione normale e compattata noterete che alcuni in particolare perdono molto di qualita'.....

Lezione14-10c

```
;  Lezione14-10c.s - Uso della routine player6.1a per un modulo non compresso

; VERSIONE CIA! La routine P61_Music non va mai chiamata!

        SECTION Usoplay61a,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;
; Call P61_Init to initialize the playroutine
; D0 --> Timer detection (for CIA-version)
; A0 --> Address to the module
; A1 --> Address to samples/0
; A2 --> Address to sample buffer
```

```
; D0 <-- 0 if succeeded
; A6 <-- $DFF000
;

        movem.l d0-d7/a0-a6,-(SP)
        moveq   #0,d0           ; Timer Detection: Autodetect
;       moveq   #1,d0           ; Timer Detection: PAL
;       moveq   #2,d0           ; Timer Detection: NTSC
        lea     P61_data,a0     ; Indirizzo del modulo in a0
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
        sub.l   a2,a2           ; no samples -> modulo non compattato
        bsr.w   P61_Init
        movem.l (SP)+,d0-d7/a0-a6

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.w  #$e000,$9a(a5)          ; INTENA - Abilito Master and lev6
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$08000,d2      ; linea da aspettare = $80
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta

; versione CIA, non occorre chiamare P61_music...

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

;
; Call P61_End to stop the music
;   A6 --> Customchip baseaddress ($DFF000)
;               Uses D0/D1/A0/A1/A3
;

        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_End

        rts


*****************************************************************************
*                The Player 6.1A for Asm-One 1.09 and later                 *
*****************************************************************************

fade  = 0       ;0 = Normal, NO master volume control possible
```

```
                ;1 = Use master volume (P61_Master)

jump = 0        ;0 = do NOT include position jump code (P61_SetPosition)
                ;1 = Include

system = 0      ;0 = killer
                ;1 = friendly

CIA = 1         ;0 = CIA disabled
                ;1 = CIA enabled

exec = 1        ;0 = ExecBase destroyed
                ;1 = ExecBase valid

opt020 = 0      ;0 = MC680x0 code
                ;1 = MC68020+ or better

use = $2009559  ; Usecode (mettete il valore dato dal p61con al salvataggio
                ; diverso per ogni modulo!)

*****************************************************************************
        include "play.s"        ; La routine vera e propria!
*****************************************************************************


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $180,$003       ; color0 nero
        dc.W    $FFFF,$FFFE     ; fine della copperlist

*****************************************************************************
;       Modulo musicale convertito in formato P61, non compresso
*****************************************************************************

        Section modulozzo,data_C

; Il modulo e' di DreamFish. Originale 42684, convertito 31628 (non packed!)

P61_data:
        incbin  "P61.technochild"       ; non compresso, solo convertito.

        end
```

Questo e' un esempio di come usare l'opzione che usa interamente i timer CIA per temporizzare... per cui non occorre chiamare P61_Music ad ogni vertical blank. Comunque, vi consiglio di usare il sistema normale di chiamata a P61_Music ad ogni frame, almeno sapete con certezza "quando" viene eseguita.

Lezione14-10d

```
;  Lezione14-10d.s - Uso della routine player6.1a per un modulo compresso

; VERSIONE CIA! La routine P61_Music non va mai chiamata!
```

```
        SECTION Usoplay61a,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;
; Call P61_Init to initialize the playroutine
; D0 --> Timer detection (for CIA-version)
; A0 --> Address to the module
; A1 --> Address to samples/0
; A2 --> Address to sample buffer
; D0 <-- 0 if succeeded
; A6 <-- $DFF000
;

        movem.l d0-d7/a0-a6,-(SP)
        moveq   #0,d0           ; Timer Detection: Autodetect
;       moveq   #1,d0           ; Timer Detection: PAL
;       moveq   #2,d0           ; Timer Detection: NTSC
        lea     P61_data,a0     ; Indirizzo del modulo in a0
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
        lea     samples,a2      ; modulo compattato! Buffer destinazione per
                                ; i samples (in chip ram) da indicare!
        bsr.w   P61_Init        ; Nota: impiega alcuni secondi per decompress!
        movem.l (SP)+,d0-d7/a0-a6

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.w  #$e000,$9a(a5)          ; INTENA - Abilito Master and lev6
        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$08000,d2      ; linea da aspettare = $80
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta
```

```
; versione CIA, non occorre chiamare P61_music...

        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse


        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_End

        rts


*****************************************************************************
*                The Player 6.1A for Asm-One 1.09 and later                 *
*****************************************************************************

fade  = 0       ;0 = Normal, NO master volume control possible
                ;1 = Use master volume (P61_Master)

jump = 0        ;0 = do NOT include position jump code (P61_SetPosition)
                ;1 = Include

system = 0      ;0 = killer
                ;1 = friendly

CIA = 1         ;0 = CIA disabled
                ;1 = CIA enabled

exec = 1        ;0 = ExecBase destroyed
                ;1 = ExecBase valid

opt020 = 0      ;0 = MC680x0 code
                ;1 = MC68020+ or better

use = $b55a     ; Usecode (mettete il valore dato dal p61con al salvataggio
                ; diverso per ogni modulo!)

*****************************************************************************
        include "play.s"        ; La routine vera e propria!
*****************************************************************************


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $180,$003       ; color0 nero
        dc.W    $FFFF,$FFFE     ; fine della copperlist

*****************************************************************************
;       Modulo musicale convertito in formato P61, COMPRESSO! (opzione pack!)
*****************************************************************************

        Section modulozzo,data  ; Non occorre sia in chip ram, perche' e'
                                ; compresso e sara' scompattato altrove!

; Il modulo e' di Jester/Sanity. Originale 153676, packed 71950
```

```
P61_data:
        incbin  "P61.stardust"  ; Compresso, (opzione PACK SAMPLES), per cui
                                ; si puo' mettere anche in fast ram: sara'
                                ; usato per scompattare i samples nel buffer
                                ; samples, e non sara' "suonato" direttamente,
                                ; quindi non dovra' passare per i canali DMA
                                ; audio, ma solo dalla routine di depack del
                                ; processore. Quindi, basta un DATA (non _C!)


*****************************************************************************
;       Dove saranno scompattati i samples (section bss in chip ram!)
*****************************************************************************

        section smp,bss_c

samples:
        ds.b    132112  ; lunghezza riportata dal p61con

        end
```

Uso della routine CIA version con un modulo compattato... giusto per amore
degli esempi.

Lezione14-10e

```
;  Lezione14-10e.s - Uso della routine player6.1a per un modulo non compresso

; Routine P61_Music chiamata da interrupt VERTB ($6c) livello3

; Inoltre e' abilitato il controllo del master volume: opzione fade=1

        SECTION Usoplay61a,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; solo copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;
; Call P61_Init to initialize the playroutine
; D0 --> Timer detection (for CIA-version)
; A0 --> Address to the module
; A1 --> Address to samples/0
; A2 --> Address to sample buffer
; D0 <-- 0 if succeeded
; A6 <-- $DFF000
;

        movem.l d0-d7/a0-a6,-(SP)
        lea     P61_data,a0     ; Indirizzo del modulo in a0
```

```
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
        sub.l   a2,a2           ; no samples -> modulo non compattato
        bsr.w   P61_Init
        movem.l (SP)+,d0-d7/a0-a6

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  BaseVBR(PC),a0
        move.l  #Myint6c,$6c(a0)        ; metto la mia routine interrupt

        move.w  #$e020,$9a(a5)          ; INTENA - Abilito Master and lev6
                                        ; e VERTB (lev3).

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

        clr.w   P61_Master      ; volume=0

mouse:
        bsr.s   SpettaBlank
        bsr.s   AlzaVolume
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

mouse2:
        bsr.s   SpettaBlank
        bsr.s   AbbassaVolume
        btst    #2,$dff016      ; mouse premuto?
        bne.s   mouse2

;
; Call P61_End to stop the music
;   A6 --> Customchip baseaddress ($DFF000)
;               Uses D0/D1/A0/A1/A3
;

        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_End

        rts

*****************************************************************************
; Routine che aspetta il Vblank
*****************************************************************************


SpettaBlank:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$10000,d2      ; linea da aspettare = $100
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $100
        BNE.S   Waity1
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
```

```
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $12c
        BEQ.S   Aspetta
        rts

*****************************************************************************
; Semplici routines che controllano la label P61_Master                      *
*****************************************************************************

AbbassaVolume:
        tst.w   P61_Master      ; Minimo volume raggiunto?
        beq.s   NonAbbassare
        subq.w  #1,P61_Master
NonAbbassare:
        rts

AlzaVolume:
        cmp.w   #64,P61_Master  ; Massimo volume raggiunto?
        beq.s   NonAlzare
        addq.w  #1,P61_Master
NonAlzare:
        rts

*****************************************************************************
*               Routine in interrupt livello 3 ($6c)                        *
*****************************************************************************

MyInt6c:
        btst    #5,$dff01f      ; INTREQR - int VERTB?
        beq.s   noint           ; se no, esci!

        movem.l d0-d7/a0-a6,-(SP)
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_Music
        movem.l (SP)+,d0-d7/a0-a6

noint:
        move.w  #$70,$dff09c    ; INTENAR
        rte

*****************************************************************************
*                The Player 6.1A for Asm-One 1.09 and later                 *
*****************************************************************************

fade  = 1       ;0 = Normal, NO master volume control possible
                ;1 = Use master volume (P61_Master)

jump = 0        ;0 = do NOT include position jump code (P61_SetPosition)
                ;1 = Include

system = 0      ;0 = killer
                ;1 = friendly

CIA = 0         ;0 = CIA disabled
                ;1 = CIA enabled

exec = 1        ;0 = ExecBase destroyed
                ;1 = ExecBase valid

opt020 = 0      ;0 = MC680x0 code
                ;1 = MC68020+ or better
```

```
use = $2009559  ; Usecode (mettete il valore dato dal p61con al salvataggio
                ; diverso per ogni modulo!)

*****************************************************************************
        include "play.s"        ; La routine vera e propria!
*****************************************************************************


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $180,$003       ; color0 nero
        dc.W    $FFFF,$FFFE     ; fine della copperlist

*****************************************************************************
;       Modulo musicale convertito in formato P61, non compresso
*****************************************************************************

        Section modulozzo,data_C

; Il modulo e' di DreamFish. Originale 42684, convertito 31628 (non packed!)

P61_data:
        incbin  "P61.technochild"       ; non compresso, solo convertito.

        end
```

L'utilita' del "fade" audio e' evidente... alla fine di un livello del vostro gioco, o all'uscita della demo, eccetera.
Basta attivare l'equate "fade", e agire sulla label "P61_Master".
Giusto per variare, qua la routine viene eseguita in interrupt VERTB ($6c).

Lezione14-10f

```
;  Lezione14-10f.s - Uso della routine player6.1a per un modulo compresso

; Routine P61_Music chiamata da interrupt VERTB ($6c) livello3

; Inoltre e' abilitata la routine di salto ai vari punti del modulo. Per
; fare cio' basta settare jump = 1, e chiamare la routine P61_SetPosition
; con la posizione in d0.l

; Premere alternativamente i tasti desto e sinistro, ma attenzione al fatto
; che i cambiamenti avvengono alla fine del pattern, non subito!!!

        SECTION Usoplay61a,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s" ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001111000000       ; solo copper DMA
```

```
WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;
; Call P61_Init to initialize the playroutine
; D0 --> Timer detection (for CIA-version)
; A0 --> Address to the module
; A1 --> Address to samples/0
; A2 --> Address to sample buffer
; D0 <-- 0 if succeeded
; A6 <-- $DFF000
;

        movem.l d0-d7/a0-a6,-(SP)
        lea     P61_data,a0     ; Indirizzo del modulo in a0
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        sub.l   a1,a1           ; I samples non sono a parte, mettiamo zero
        lea     samples,a2      ; modulo compattato! Buffer destinazione per
                                ; i samples (in chip ram) da indicare!
        bsr.w   P61_Init        ; Nota: impiega alcuni secondi per decompress!
        movem.l (SP)+,d0-d7/a0-a6

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
                                        ; e sprites.

        move.l  BaseVBR(PC),a0
        move.l  #Myint6c,$6c(a0)        ; metto la mia routine interrupt

        move.w  #$e020,$9a(a5)          ; INTENA - Abilito Master and lev6
                                        ; e VERTB (lev3).

        move.l  #COPPERLIST,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Disattiva l'AGA
        move.w  #$c00,$106(a5)          ; Disattiva l'AGA
        move.w  #$11,$10c(a5)           ; Disattiva l'AGA

mouse:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse

        clr.w   ModPos          ; riparti da capo
        st.b    CambiaPos

mouse2:
        btst    #2,$dff016      ; mouse premuto?
        bne.s   mouse2

        move.w  #16,ModPos      ; vai alla pos 16
        st.b    CambiaPos

mouse3:
        btst    #6,$bfe001      ; mouse premuto?
        bne.s   mouse3

        move.w  #30,ModPos      ; Vai alla posizione 30 (in questo modulo e'
        st.b    CambiaPos       ; l'ultima).

mouse4:
```

```
        btst    #2,$dff016      ; mouse premuto?
        bne.s   mouse4

;
; Call P61_End to stop the music
;   A6 --> Customchip baseaddress ($DFF000)
;               Uses D0/D1/A0/A1/A3
;

        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        bsr.w   P61_End

        rts

*****************************************************************************
*               Routine in interrupt livello 3 ($6c)                        *
*****************************************************************************

MyInt6c:
        btst    #5,$dff01f      ; INTREQR - int VERTB?
        beq.s   noint           ; se no, esci!

        movem.l d0-d7/a0-a6,-(SP)
        lea     $dff000,a6      ; Ricordiamoci il $dff000 in a6!
        tst.b   CambiaPos       ; dobbiamo saltare pos?
        beq.s   NonCambiarPos
        cmp.w   #63,P61_Crow    ; siamo all'ultima riga del pattern?
        bne.s   NonCambiarPos   ; se non ancora, non ripartire da capo!
        clr.b   CambiaPos
        moveq   #0,d0
        move.w  ModPos(PC),d0   ; a quale pos. saltiamo?
        bsr.w   P61_SetPosition ; cambiamo posizione
NonCambiarPos:
        bsr.w   P61_Music               ; suoniamo
        movem.l (SP)+,d0-d7/a0-a6
noint:
        move.w  #$70,$dff09c    ; INTENAR
        rte

ModPos:
        dc.w    0
CambiaPos:
        dc.w    0

*****************************************************************************
*                The Player 6.1A for Asm-One 1.09 and later                 *
*****************************************************************************

fade  = 0       ;0 = Normal, NO master volume control possible
                ;1 = Use master volume (P61_Master)

jump = 1        ;0 = do NOT include position jump code (P61_SetPosition)
                ;1 = Include

system = 0      ;0 = killer
                ;1 = friendly

CIA = 0         ;0 = CIA disabled
                ;1 = CIA enabled

exec = 1        ;0 = ExecBase destroyed
                ;1 = ExecBase valid
```

```
opt020 = 0      ;0 = MC680x0 code
                ;1 = MC68020+ or better

use = $b55a     ; Usecode (mettete il valore dato dal p61con al salvataggio
                ; diverso per ogni modulo!)

*****************************************************************************
        include "play.s"        ; La routine vera e propria!
*****************************************************************************


*****************************************************************************
;       Copperlist
*****************************************************************************

        SECTION COP,DATA_C

COPPERLIST:
        dc.w    $100,$200       ; bplcon0 - no bitplanes
        DC.W    $180,$003       ; color0 nero
        dc.W    $FFFF,$FFFE     ; fine della copperlist

*****************************************************************************
;       Modulo musicale convertito in formato P61, COMPRESSO! (opzione pack!)
*****************************************************************************

        Section modulozzo,data  ; Non occorre sia in chip ram, perche' e'
                                ; compresso e sara' scompattato altrove!

; Il modulo e' di Jester/Sanity. Originale 153676, packed 71950

P61_data:
        incbin  "P61.stardust"  ; Compresso, (opzione PACK SAMPLES), per cui
                                ; si puo' mettere anche in fast ram: sara'
                                ; usato per scompattare i samples nel buffer
                                ; samples, e non sara' "suonato" direttamente,
                                ; quindi non dovra' passare per i canali DMA
                                ; audio, ma solo dalla routine di depack del
                                ; processore. Quindi, basta un DATA (non _C!)


*****************************************************************************
;       Dove saranno scompattati i samples (section bss in chip ram!)
*****************************************************************************

        section smp,bss_c

samples:
        ds.b    132112  ; lunghezza riportata dal p61con

        end

Fate attenzione nell'uso della routine che salta qua' e la' nel modulo!
Innanzitutto, se saltate nel mezzo di un pattern, va tutto fuori sincronia,
non so se per un bug del player o per altro. Quindi occorre attendere la
fine del pattern corrente prima di saltare ad un altro. Si puo' sapere in
qualunque momento a che punto del modulo siamo leggendo queste 3 variabili:

        P61_Pos:        Current song position
        P61_Patt:       Current pattern
        P61_CRow:       Current row in pattern
```

```
L'utilita' di questa routine si puo' trovare soltanto nel caso in cui si faccia
un modulo in cui volontariamente non si raggiunge mai una data posizione, a
cui si debba saltare con questa routine. Per esempio, per un gioco si puo'
fare una musica "tranquilla" di base, che loopa sempre, ed occupa le prime
40 posizioni. Pero', alle posizioni dalla 40 alla 50 c'e' un altro motivo,
piu' drammatico, a cui non si puo' accedere se non ci si salta.
Ecco allora il nostro metto che se ne va per il mondo, con la musica
spensierata di sfondo... poi trova il cattivo, e saltiamo all'altro motivetto,
che loopa per conto suo... ucciso il mostro, torniamo alla musichetta
tranquillissima.
```

CAPITOLO 29

LEZIONE 15

29.1 Lezione15a

```
; Lezione15a.s          Sfumatura copper AGA con utilizzo della palette 24bit.
;                       Sotto si nota la differenza con quella a 12bit.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001010000000       ; copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************
```

```
        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $100,$201       ; no bitplanes (bit 1 abilitato, pero'!)

        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$005       ; Color0 - nibble alti
                                ; (I nibble bassi li lasciamo a zero...)

        dc.w    $5f07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$010       ; Color0 - nibble bassi

        dc.w    $6007,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$030       ; Color0 - nibble bassi

        dc.w    $6107,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$050       ; Color0 - nibble bassi

        dc.w    $6207,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$070       ; Color0 - nibble bassi

        dc.w    $6307,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$090       ; Color0 - nibble bassi

        dc.w    $6407,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0b0       ; Color0 - nibble bassi

        dc.w    $6507,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0d0       ; Color0 - nibble bassi

        dc.w    $6607,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
```

```
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0f0       ; Color0 - nibble bassi

        dc.w    $6707,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$010       ; Color0 - nibble bassi

        dc.w    $6807,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$030       ; Color0 - nibble bassi

        dc.w    $6907,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$050       ; Color0 - nibble bassi

        dc.w    $6a07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$070       ; Color0 - nibble bassi

        dc.w    $6b07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$090       ; Color0 - nibble bassi

        dc.w    $6c07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0b0       ; Color0 - nibble bassi

        dc.w    $6d07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0d0       ; Color0 - nibble bassi

        dc.w    $6e07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0f0       ; Color0 - nibble bassi

        dc.w    $6f07,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$010       ; Color0 - nibble bassi

        dc.w    $7007,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$030       ; Color0 - nibble bassi
```

```
        dc.w    $7107,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$050       ; Color0 - nibble bassi

        dc.w    $7207,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$070       ; Color0 - nibble bassi

        dc.w    $7307,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$090       ; Color0 - nibble bassi

        dc.w    $7407,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0b0       ; Color0 - nibble bassi

        dc.w    $7507,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0d0       ; Color0 - nibble bassi

        dc.w    $7607,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$0f0       ; Color0 - nibble bassi

        dc.w    $7707,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$030       ; Color0 - nibble alti
        dc.w    $106,$e00       ; SELEZIONA NIBBLE BASSI
        dc.w    $180,$010       ; Color0 - nibble bassi

; Ora mettiamo a confronto con la palette "standard" ECS/OCS:

        dc.w    $7907,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti

        dc.w    $8007,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$010       ; Color0 - nibble alti

        dc.w    $8807,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$020       ; Color0 - nibble alti

        dc.w    $9007,$fffe     ; Wait
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$030       ; Color0 - nibble alti

        dc.w    $9807,$fffe     ; Wait
```

```
        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$005       ; Color0 - nibble alti

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

        end
```

```
Si nota la differenza, vero?? AGA rulez!
Se notate, la sfumatura segue questo andamento:

  Diviso per nibble       Originale a 24 bit

          RGB   rgb               RrGgBb
        $0000,$0000     -> ossia $000000
        $0000,$0010     -> ossia $000100
        $0000,$0030     -> ossia $000300
        $0000,$0050     -> ossia $000500
        $0000,$0070     -> ossia $000700
        $0000,$0090     -> ossia $000900
        $0000,$00B0     -> ossia $000b00
        $0000,$00D0     -> ossia $000d00
        $0000,$00F0     -> ossia $000f00
        $0010,$0010     -> ossia $001100
        $0010,$0030     -> ossia $001300
        $0010,$0050     -> ossia $001500
        $0010,$0070     -> ossia $001700
        $0010,$0090     -> ossia $001900
        $0010,$00B0     -> ossia $001b00
        $0010,$00D0     -> ossia $001d00
        $0010,$00F0     -> ossia $001f00
        $0020,$0010     -> ossia $002100
        $0020,$0030     -> ossia $002300
        $0020,$0050     -> ossia $002500
        $0020,$0070     -> ossia $002700
        $0020,$0090     -> ossia $002900
        $0020,$00B0     -> ossia $002b00
        $0020,$00D0     -> ossia $002d00
        $0020,$00F0     -> ossia $002f00
        $0030,$0010     -> ossia $003100
        ...

Fare una sfumatura AGA e' lungo manualmente, conviene farsi una routine che
le crei!
```

29.2 Lezione15b

```
; Lezione15b.s          Sfumatura copper AGA con utilizzo della palette 24bit.
;                       Usiamo una routine per fare la sfumatura.
;                       Tasti destro e sinistro per uscire

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
```

Figura 29.1: Lezione 15a

```
DMASET  EQU     %1000001010000000       ; copper DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
        move.w  #200-1,d7       ; Numero linee: 200!
        bsr.w   FaiAGACopR      ; Fai una sfumatura ROSSA

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP1:
        BTST.b  #6,$BFE001      ; tasto sin mouse?
        BNE.S   LOOP1


        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
        move.w  #200-1,d7       ; Numero linee: 200!
        bsr.w   FaiAGACopG      ; Fai una sfumatura GREEN (verde)

LOOP2:
        BTST.b  #2,$16(a5)      ; tasto destro mouse?
        BNE.S   LOOP2

        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
```

```
        move.w  #200-1,d7       ; Numero linee: 200!
        bsr.w   FaiAGACopB      ; Fai una sfumatura BLU

LOOP3:
        BTST.b  #6,$BFE001      ; tasto sin mouse?
        BNE.S   LOOP3

        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
        move.w  #150-1,d7       ; Numero linee: 150!
        bsr.s   FaiAGACopG      ; Fai una sfumatura GREEN (verde)

LOOP4:
        BTST.b  #2,$16(a5)      ; tasto destro mouse?
        BNE.S   LOOP4

        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
        move.w  #60-1,d7        ; Numero linee: 200!
        bsr.w   FaiAGACopR      ; Fai una sfumatura ROSSA

LOOP5:
        BTST.b  #6,$BFE001      ; tasto sin mouse?
        BNE.S   LOOP5

        RTS

;*****************************************************************************
; Routine che crea sfumature aga ROSSE:
;
; d1 = Prima linea da aspettare (Wait, ad es: $2c07fffe per linea Y=$2c)
; d5 = inizio tonalita' ($00-$ff)
; d7 = Numero linee da fare
;*****************************************************************************

FaiAgaCopR:
        lea     AgaCopEff1,a0
        move.l  #$01060c00,d4   ; BplCon3 - nibble alti
        move.l  #$01060e00,d3   ; BplCon3 - nibble bassi
        move.w  #$180,d2        ; Registro Color0
FaiAGALoopR:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        move.l  d4,(a0)+        ; BplCon3 - selez. nibble alti
        move.w  d2,(a0)+        ; Registro Color0
        addq.b  #1,d5           ; "Illumina" leggermente il colore $Gg
        move.w  d5,d6           ; Copialo in d6
        and.w   #%11110000,d6   ; Selez. solo il nibble ALTO
        lsl.w   #4,d6           ; Alla posizione giusta, ossia al RED ($Rxx)
        move.w  d6,(a0)+        ; Valore del Color0 (nib alti)
        move.l  d3,(a0)+        ; BplCon3 - selez. nibble bassi
        move.w  d2,(a0)+        ; Registro Color0
        move.w  d5,d6           ; Colore $xx in d6
        and.w   #%00001111,d6   ; Selez. solo i nibble bassi
        lsl.w   #8,d6           ; Spostali alla posizione del rosso
        move.w  d6,(a0)+        ; Metti il colore in copperlist (nibble bassi)
        dbra    d7,FaiAGALoopR
        rts

;*****************************************************************************
; Routine che crea sfumature aga VERDI:
;
```

```
; d1 = Prima linea da aspettare (Wait, ad es: $2c07fffe per linea Y=$2c)
; d5 = inizio tonalita' ($00-$ff)
; d7 = Numero linee da fare
;***************************************************************************

FaiAgaCopG:
        lea     AgaCopEff1,a0
        move.l  #$01060c00,d4   ; BplCon3 - nibble alti
        move.l  #$01060e00,d3   ; BplCon3 - nibble bassi
        move.w  #$180,d2        ; Registro Color0
FaiAGALoopG:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        move.l  d4,(a0)+        ; BplCon3 - selez. nibble alti
        move.w  d2,(a0)+        ; Registro Color0
        addq.b  #1,d5           ; "Illumina" leggermente il colore $Gg
        move.w  d5,d6           ; Copialo in d6
        and.w   #%11110000,d6   ; Selez. solo il nibble ALTO (e' gia' alla
                                ; posizione giusta, ossia al GREEN $xGx)
        move.w  d6,(a0)+        ; Valore del Color0 (nib alti)
        move.l  d3,(a0)+        ; BplCon3 - selez. nibble bassi
        move.w  d2,(a0)+        ; Registro Color0
        move.w  d5,d6           ; Colore $xx in d6
        and.w   #%00001111,d6   ; Selez. solo i nibble bassi
        lsl.w   #4,d6           ; Spostali alla posizione del verde
        move.w  d6,(a0)+        ; Metti il colore in copperlist (nibble bassi)
        dbra    d7,FaiAGALoopG
        rts

;***************************************************************************
; Routine che crea sfumature aga BLU:
;
; d1 = Prima linea da aspettare (Wait, ad es: $2c07fffe per linea Y=$2c)
; d5 = inizio tonalita' ($00-$ff)
; d7 = Numero linee da fare
;***************************************************************************

FaiAgaCopB:
        lea     AgaCopEff1,a0
        move.l  #$01060c00,d4   ; BplCon3 - nibble alti
        move.l  #$01060e00,d3   ; BplCon3 - nibble bassi
        move.w  #$180,d2        ; Registro Color0
FaiAGALoopB:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        move.l  d4,(a0)+        ; BplCon3 - selez. nibble alti
        move.w  d2,(a0)+        ; Registro Color0
        addq.b  #1,d5           ; "Illumina" leggermente il colore $Gg
        move.w  d5,d6           ; Copialo in d6
        and.w   #%11110000,d6   ; Selez. solo il nibble ALTO
        lsr.w   #4,d6           ; Alla posizione giusta, ossia al BLU $xxB)
        move.w  d6,(a0)+        ; Valore del Color0 (nib alti)
        move.l  d3,(a0)+        ; BplCon3 - selez. nibble bassi
        move.w  d2,(a0)+        ; Registro Color0
        move.w  d5,d6           ; Colore $xx in d6
        and.w   #%00001111,d6   ; Selez. solo i nibble bassi - posizione $xxB
        move.w  d6,(a0)+        ; Metti il colore in copperlist (nibble bassi)
        dbra    d7,FaiAGALoopB
        rts

;***************************************************************************
;*                            COPPERLIST                                   *
```

```
;******************************************************************************

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $100,$201       ; no bitplanes (bit 1 abilitato, pero'!)

        dc.w    $106,$c00       ; SELEZIONA NIBBLE ALTI
        dc.w    $180,$000       ; Color0 - nibble alti
                                ; (I nibble bassi li lasciamo a zero...)

AgaCopEff1:
        dcb.l   200*5           ; Ossia: 200 linee * 5 long:
                                ; 1 per il wait,
                                ; 1 per il bplcon3
                                ; 1 per color0 (nib alti)
                                ; 1 per il bplcon3
                                ; 1 per color0 (nib bassi)

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

        end

Conveniva proprio farsi una routine per questa sfumatura. Vi immaginate quante
linee avremmo dovuto scrivere???
```

Figura 29.2: Lezione 15b

29.3 Lezione15c

```
; Lezione15c.s  Visualizzazione di una figura lowres a 256 colori (8 planes)

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7                 ; num bitplanes -1
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        add.l   #40*256,d0              ; lunghezza bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        BTST.b  #6,$BFE001
        BNE.S   LOOP
        RTS

;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
```

```
        dc.w    $10a,0          ; Bpl2Mod

                ; 5432109876543210
        dc.w    $100,%0000001000010001  ; 8 bitplane LOWRES 320x256

        dc.w    $1fc,0          ; Burst mode azzerato

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Colori salvati dall'iffconverter. Da PicCon occorre settare nel menu' dei
; Settings/Paletteformat il "pulsante" COPPERLIST, il che salva una copperlist
; con i registri $106 e $180,$182 ecc. anziche' i soli valori. Inoltre occorre
; salcare come sorgente in "dc.w" anziche' in binario, e questo si puo' fare
; settando il menu' Project/Save data as../*ASM source*, anziche' bynary.
; Per l'iffconv di Yragael basta salvare come COPPER, ma la lista viene piu'
; lunga perche' mette un registro per linea. L'agaconv salva i registri colore
; senza i $106, per cui si dovrebbe metterli a mano, vi consiglio di salvare
; la copperlist con il PicCon o l'Iffconv. (nota: i registri $106 non hanno i
; bit 10 e 11 settati, per cui sono $106,$000 - $106,$200 - $106,$2000 ecc,
; la funzione e' la stessa. Una volta settata la palette potete cambiare dei
; bit del $dff106 (BPLCON3) a vostro piacimento.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI

        dc.w    $180,$010,$182,$100,$184,$011,$186,$110 ; nibble alti dei
        dc.w    $188,$020,$18a,$120,$18c,$022,$18e,$210 ; colori 0-31
        dc.w    $190,$121,$192,$130,$194,$311,$196,$320
        dc.w    $198,$131,$19a,$230,$19c,$023,$19e,$321
        dc.w    $1a0,$332,$1a2,$240,$1a4,$133,$1a6,$420
        dc.w    $1a8,$421,$1aa,$233,$1ac,$024,$1ae,$340
        dc.w    $1b0,$422,$1b2,$430,$1b4,$250,$1b6,$520
        dc.w    $1b8,$432,$1ba,$333,$1bc,$034,$1be,$234

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI

        dc.w    $180,$214,$182,$5f4,$184,$015,$186,$969 ; nibble bassi dei
        dc.w    $188,$926,$18a,$877,$18c,$540,$18e,$f56 ; colori 0-31
        dc.w    $190,$79c,$192,$f18,$194,$463,$196,$53b
        dc.w    $198,$c3c,$19a,$c49,$19c,$00d,$19e,$54f
        dc.w    $1a0,$240,$1a2,$17a,$1a4,$097,$1a6,$a07
        dc.w    $1a8,$70b,$1aa,$4a2,$1ac,$0aa,$1ae,$54d
        dc.w    $1b0,$982,$1b2,$4ca,$1b4,$15a,$1b6,$48b
        dc.w    $1b8,$841,$1ba,$a91,$1bc,$76f,$1be,$3e5

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        dc.w    $180,$143,$182,$350,$184,$035,$186,$253 ; nibble alti dei
        dc.w    $188,$441,$18a,$530,$18c,$343,$18e,$532 ; colori 32-63
        dc.w    $190,$533,$192,$720,$194,$451,$196,$046
        dc.w    $198,$542,$19a,$541,$19c,$453,$19e,$146
        dc.w    $1a0,$721,$1a2,$364,$1a4,$543,$1a6,$346
        dc.w    $1a8,$561,$1aa,$730,$1ac,$742,$1ae,$651
        dc.w    $1b0,$454,$1b2,$733,$1b4,$156,$1b6,$821
        dc.w    $1b8,$266,$1ba,$652,$1bc,$741,$1be,$047
```

```
DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

dc.w    $180,$ddc,$182,$29c,$184,$13d,$186,$70d ; nibble bassi, ecc.
dc.w    $188,$8f1,$18a,$aee,$18c,$5ec,$18e,$c91
dc.w    $190,$941,$192,$03e,$194,$cb2,$196,$606
dc.w    $198,$b4c,$19a,$dd1,$19c,$6a0,$19e,$037
dc.w    $1a0,$3e1,$1a2,$113,$1a4,$b88,$1a6,$262
dc.w    $1a8,$074,$1aa,$6bf,$1ac,$003,$1ae,$571
dc.w    $1b0,$dd4,$1b2,$3f1,$1b4,$93e,$1b6,$a02
dc.w    $1b8,$b21,$1ba,$c5e,$1bc,$8f9,$1be,$dcd

DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

dc.w    $180,$742,$182,$821,$184,$761,$186,$662
dc.w    $188,$456,$18a,$830,$18c,$158,$18e,$754
dc.w    $190,$761,$192,$931,$194,$566,$196,$841
dc.w    $198,$158,$19a,$850,$19c,$762,$19e,$852
dc.w    $1a0,$582,$1a2,$764,$1a4,$861,$1a6,$852
dc.w    $1a8,$765,$1aa,$854,$1ac,$b21,$1ae,$169
dc.w    $1b0,$a31,$1b2,$a21,$1b4,$871,$1b6,$269
dc.w    $1b8,$a52,$1ba,$782,$1bc,$a51,$1be,$963

DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

dc.w    $180,$99e,$182,$f73,$184,$127,$186,$c1b
dc.w    $188,$5e1,$18a,$f7f,$18c,$b01,$18e,$085
dc.w    $190,$1e6,$192,$4c7,$194,$143,$196,$bef
dc.w    $198,$ef3,$19a,$f1f,$19c,$4dd,$19e,$91e
dc.w    $1a0,$8a4,$1a2,$7a5,$1a4,$c3d,$1a6,$eef
dc.w    $1a8,$23d,$1aa,$c64,$1ac,$005,$1ae,$d71
dc.w    $1b0,$ac5,$1b2,$fd4,$1b4,$d49,$1b6,$bd0
dc.w    $1b8,$15d,$1ba,$8aa,$1bc,$2f3,$1be,$3d1

DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

dc.w    $180,$a41,$182,$677,$184,$379,$186,$b21
dc.w    $188,$881,$18a,$a62,$18c,$964,$18e,$a61
dc.w    $190,$586,$192,$875,$194,$a64,$196,$279
dc.w    $198,$a73,$19a,$c21,$19c,$b61,$19e,$883
dc.w    $1a0,$885,$1a2,$c41,$1a4,$37a,$1a6,$877
dc.w    $1a8,$b62,$1aa,$b61,$1ac,$a74,$1ae,$a75
dc.w    $1b0,$a82,$1b2,$886,$1b4,$a84,$1b6,$48a
dc.w    $1b8,$a92,$1ba,$a76,$1bc,$c61,$1be,$b73

DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

dc.w    $180,$ce3,$182,$e01,$184,$a12,$186,$da6
dc.w    $188,$7be,$18a,$17e,$18c,$2f7,$18e,$4e3
dc.w    $190,$de7,$192,$d04,$194,$056,$196,$f3f
dc.w    $198,$331,$19a,$9b6,$19c,$610,$19e,$ebb
dc.w    $1a0,$1d5,$1a2,$703,$1a4,$d94,$1a6,$c13
dc.w    $1a8,$3eb,$1aa,$ac1,$1ac,$d24,$1ae,$723
dc.w    $1b0,$5b7,$1b2,$fc0,$1b4,$544,$1b6,$494
dc.w    $1b8,$570,$1ba,$761,$1bc,$d24,$1be,$a6e

DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

dc.w    $180,$a84,$182,$d41,$184,$a85,$186,$48b
dc.w    $188,$b75,$18a,$68a,$18c,$c72,$18e,$8a6
dc.w    $190,$d61,$192,$c82,$194,$b84,$196,$c81
dc.w    $198,$a86,$19a,$d62,$19c,$a95,$19e,$889
```

```
dc.w    $1a0,$59b,$1a2,$e61,$1a4,$c92,$1a6,$c84
dc.w    $1a8,$b95,$1aa,$d63,$1ac,$c94,$1ae,$ca2
dc.w    $1b0,$a97,$1b2,$d74,$1b4,$59c,$1b6,$899
dc.w    $1b8,$c95,$1ba,$b97,$1bc,$e62,$1be,$e71

DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

dc.w    $180,$7f1,$182,$b14,$184,$4b5,$186,$590
dc.w    $188,$874,$18a,$0f2,$18c,$978,$18e,$b28
dc.w    $190,$4d0,$192,$548,$194,$89a,$196,$864
dc.w    $198,$4ce,$19a,$8b2,$19c,$5e9,$19e,$9f7
dc.w    $1a0,$a43,$1a2,$427,$1a4,$547,$1a6,$990
dc.w    $1a8,$929,$1aa,$fa9,$1ac,$343,$1ae,$205
dc.w    $1b0,$7a4,$1b2,$a3c,$1b4,$551,$1b6,$cf5
dc.w    $1b8,$716,$1ba,$921,$1bc,$f37,$1be,$a55

DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

dc.w    $180,$d81,$182,$a98,$184,$e72,$186,$d82
dc.w    $188,$ca3,$18a,$d93,$18c,$69c,$18e,$ca5
dc.w    $190,$c97,$192,$5ac,$194,$f81,$196,$d95
dc.w    $198,$e91,$19a,$da4,$19c,$f82,$19e,$ea2
dc.w    $1a0,$ca7,$1a2,$aaa,$1a4,$7ac,$1a6,$e94
dc.w    $1a8,$8ac,$1aa,$ca6,$1ac,$f92,$1ae,$ca8
dc.w    $1b0,$f91,$1b2,$e95,$1b4,$da6,$1b6,$fa3
dc.w    $1b8,$cb7,$1ba,$ca9,$1bc,$eb2,$1be,$eb4

DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

dc.w    $180,$eba,$182,$898,$184,$b63,$186,$fd9
dc.w    $188,$79e,$18a,$d0f,$18c,$6a7,$18e,$55b
dc.w    $190,$770,$192,$d68,$194,$0a8,$196,$69f
dc.w    $198,$a7a,$19a,$d03,$19c,$37b,$19e,$14d
dc.w    $1a0,$a42,$1a2,$823,$1a4,$17b,$1a6,$f31
dc.w    $1a8,$066,$1aa,$aef,$1ac,$28a,$1ae,$727
dc.w    $1b0,$6ac,$1b2,$999,$1b4,$78a,$1b6,$21e
dc.w    $1b8,$c68,$1ba,$649,$1bc,$f17,$1be,$827

DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

dc.w    $180,$8bc,$182,$ea6,$184,$ca9,$186,$eb6
dc.w    $188,$bc8,$18a,$fa2,$18c,$db7,$18e,$8bd
dc.w    $190,$db9,$192,$bba,$194,$fb4,$196,$db9
dc.w    $198,$fb6,$19a,$8be,$19c,$fb7,$19e,$db9
dc.w    $1a0,$fc2,$1a2,$dc8,$1a4,$fc4,$1a6,$8ce
dc.w    $1a8,$cca,$1aa,$abd,$1ac,$eb9,$1ae,$fd3
dc.w    $1b0,$fc6,$1b2,$9be,$1b4,$fb8,$1b6,$dca
dc.w    $1b8,$fd4,$1ba,$dbb,$1bc,$cbc,$1be,$ec8

DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

dc.w    $180,$929,$182,$588,$184,$8f4,$186,$233
dc.w    $188,$c2f,$18a,$7fd,$18c,$e57,$18e,$b16
dc.w    $190,$125,$192,$e5a,$194,$73b,$196,$b11
dc.w    $198,$542,$19a,$c33,$19c,$135,$19e,$2af
dc.w    $1a0,$d4f,$1a2,$f19,$1a4,$a3a,$1a6,$703
dc.w    $1a8,$34e,$1aa,$6b4,$1ac,$2a4,$1ae,$b22
dc.w    $1b0,$397,$1b2,$8e4,$1b4,$0f5,$1b6,$664
dc.w    $1b8,$b2a,$1ba,$3aa,$1bc,$1fd,$1be,$8ed

DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI
```

```
        dc.w    $180,$fd5,$182,$ace,$184,$dda,$186,$dcb
        dc.w    $188,$fca,$18a,$fd6,$18c,$ace,$18e,$fc8
        dc.w    $190,$bce,$192,$dcb,$194,$cdd,$196,$fe6
        dc.w    $198,$fe8,$19a,$bce,$19c,$dcd,$19e,$fda
        dc.w    $1a0,$fe8,$1a2,$fda,$1a4,$ddc,$1a6,$cde
        dc.w    $1a8,$fdb,$1aa,$dde,$1ac,$eec,$1ae,$dde
        dc.w    $1b0,$fea,$1b2,$fdd,$1b4,$fec,$1b6,$dde
        dc.w    $1b8,$edd,$1ba,$ffc,$1bc,$fed,$1be,$ffe

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        dc.w    $180,$d47,$182,$339,$184,$b08,$186,$3bd
        dc.w    $188,$040,$18a,$cb6,$18c,$dd0,$18e,$7fa
        dc.w    $190,$359,$192,$fec,$194,$428,$196,$ea6
        dc.w    $198,$349,$19a,$5eb,$19c,$2e4,$19e,$508
        dc.w    $1a0,$e85,$1a2,$3f7,$1a4,$8fa,$1a6,$52c
        dc.w    $1a8,$52c,$1aa,$127,$1ac,$d12,$1ae,$947
        dc.w    $1b0,$d7a,$1b2,$225,$1b4,$920,$1b6,$2ff
        dc.w    $1b8,$9ef,$1ba,$934,$1bc,$8fa,$1be,$95e

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

        end

Per il disegno originale su muro devo ringraziare la mia ex compagna di classe
Silvia Papucci, che mi ha aiutato a dipingerlo.
Per lo scanner Andrea Scarafoni. (e' immorale scansire un proprio disegno?)
```

Lezione15c2

```
; Lezione15c2.s         - colori salvati in fondo alla pic con AGAconv.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
```

```
        MOVEQ   #8-1,D7                 ;num of bitplanes -1
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #10240,d0               ;lenght of bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        bsr.s   MettiColori

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS


; Questa routine, che e' presente anche nella mia demo WORLD OF MANGA, serve
; per leggere la palette a 24 bit salvata "attaccata" in fondo alla PIC RAW
; con l'AGAconv. Non e' ottimizzata, per rendere piu' chiaro il funzionamento.
; In pratica converte ogni colore a 24 bit, che si presenta nel formato di
; una long $00RrGgBb, dove R = nibble alto di RED, r = nibble basso di RED,
; G = nibble alto di GREEN eccetera, nel formato della copperlist aga, ossia
; in due word $0RGB con i nibble alti e $0rgb con i nibble bassi.
; Da notare che puo' essere usata anche per prendere i colori da tabelle con
; la palette a 24 bit salvata in questo formato, o come base per routines
; di ogni genere che riguardi la palette.

MettiColori:
        LEA     PICTURE+(10240*8),A0    ; indirizzo della color palette alla
                                        ; fine della figura -> in A0
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
```

```
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.


;***************************************************************************
;*                            COPPERLIST                                   *
;***************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

                ; 5432109876543210
```

```
        dc.w    $100,%0000001000010001  ; 8 bitplane LOWRES 320x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,0          ; Burst mode azzerato (per ora!)

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0
```

```
DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
```

```
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

        end
```

la PIC in RAW e' 1024 bytes piu' lunga di quanto dovrebbe essere, infatti la lunghezza di un bitplane lowres e' 10240, moltiplicato per 8 darebbe 81920, ma la pic e' lunga 82944. Quelle 256 longwords (1024 bytes=512 words=256 longs) non sono altro che la palette a 24 bit salvata in fondo alla pic, come abbiamo fatto con il kefcon per le pic non aga. La possibilita' di salvare la palette in fondo o all'inizio di una pic e' data dall'AGAconv, basta settare le sue Preferences (premendo Amiga+P) in questo modo:

```
        Palette informations    -       Behind.24
```

Lezione15c3

```
; Lezione15c3.s         - Una prima prova di fade AGA a 24bit. Precalcola
;                         il fade in una tabella.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

***************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
***************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num of bitplanes -1
POINTB:
        move.w  d0,6(a1)
        swap    d0
```

```
        move.w  d0,2(a1)
        swap    d0
        addi.l  #40*256,d0      ; lenght of bitplane
        addq.w  #8,a1
        dbra    d7,POINTB       ; Rifai D7 volte (D7=num of bitplanes)

        bsr.w   FADE256PRECALC  ; Precalcola i valori di tutto il fade, per
                                ; un totale di 256 colori.l in 256 passaggi dal
                                ; nero al colore pieno, ossia 4*256*256 bytes
                                ; di tabella: 262144 bytes precalcolati!!!

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        bsr.s   MainFadeInOut   ; Routine che sfuma dal nero al colore pieno
                                ; e viceversa.

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS


*****************************************************************************
* Questa routine fa scattare in avanti o indietro il puntatore ai colori    *
* precalcolati ActualFadeTab. Quando il fade va dal nero al colore pieno    *
* aggiunge 256 longwords al puntatore, facendolo puntare ai prossimi 256    *
* colori, ossia al prossimo fotogramma piu' scuro precalcolato. Nel caso    *
* opposto torna indietro al fotogramma precedente. La label FlagFadeInOut   *
* e' usata per controllare se il fade e' In o Out.                          *
*****************************************************************************

MainFadeInOut:
        BSR.w   MettiColori     ; Sistema i colori di questo fotogramma
                                ; prendendoli e convertendoli dalla tabella
                                ; del fade precalcolato.
        BTST.b  #1,FlagFadeInOut ; Fade In o fade Out?
        BNE.S   FadeOut
FadeIn:
        ADD.L   #256*4,ActualFadeTab    ; Prossimo fotogramma (256 colori .l)
        LEA     CTABEND,A0              ; Indirizzo fine tabella
        CMP.L   ActualFadeTab(PC),A0    ; Siamo attivati alla fine della tab
```

```
                                        ; del fade? (Colori pieni e lucenti)
        BNE.s   NonFinito
        BCHG.B  #1,FlagFadeInOut        ; Cambia la direzione del fade
FadeOut:
        SUB.L   #256*4,ActualFadeTab    ; Passo precedente (piu' scuro)
        LEA     COLORTABBY,A0           ; Indirizzo di inizio tabella
        CMP.L   ActualFadeTab(PC),A0    ; Siamo arrivati all'inizio della tab
                                        ; del fade? (Colore NERO)
        BNE.W   NonFinito
        BCHG.B  #1,FlagFadeInOut        ; cambia la direzione del fade
NonFinito:
        RTS

FlagFadeInOut:          ; Usato per decidere se FadeIn o FadeOut
        dc.w    0

ActualFadeTab:                  ; Puntatore al "fotogramma" precalcolato del
        dc.l    COLORTABBY      ; fade nella tabella COLORTABBY.


*****************************************************************************
* Questa routine precalcola tutti i colori a 24 bit del fade, e fa un bel po'*
* di lavoro, dato che deve scrivere 256*256 longwords, ossia 262144 bytes!   *
* Non e' altro che la routine di fade usata per i colori a 12 bit dell'amiga *
* normale, solo che tratta 1 byte per componente RGB, anziche' 4 bit.        *
*****************************************************************************

FADE256PRECALC:
        LEA     COLORTABBY,A1   ; DEST CALCULATED COLORS TAB
        MOVEQ   #0,D6           ; MULTIPLIER START (0-255)
FADESTEPS:
        LEA     PICTURE+(10240*8),A0    ; 24bit colors tab address
        MOVE.w  #256-1,D7               ; NUM. DI COLORI = 256

COLCALCLOOP:

;       CALCOLA IL BLU

        MOVE.L  (A0),D4                 ; READ COLOR FROM TAB
        ANDI.L  #%000011111111,D4       ; SELECT BLUE
        MULU.W  D6,D4                   ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%000011111111,D4       ; SELECT BLUE VAL
        MOVE.L  D4,D5                   ; SAVE BLUE TO D5

;       CALCOLA IL VERDE

        MOVE.L  (A0),D4                 ; READ COLOR FROM TAB
        ANDI.L  #%1111111100000000,D4   ; SELECT GREEN
        LSR.L   #8,D4                   ; -> 8 bits (so from 0 to 7)
        MULU.W  D6,D4                   ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%0000000011111111,D4   ; SELECT GREEN
        LSL.L   #8,D4                   ; <- 8 bits (so from 8 to 15)
        OR.L    D4,D5                   ; SAVE GREEN TO D5

;       CALCOLA IL BLU

        MOVE.L  (A0)+,D4                ; READ COLOR FROM TAB AND GO TO NEXT
        ANDI.L  #%111111110000000000000000,D4   ; SELECT RED
        LSR.L   #8,D4                   ; -> 8 bits (so from 8 to 15)
        LSR.L   #8,D4                   ; -> 8 bits (so from 0 to 7)
```

```
        MULU.W  D6,D4                   ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%0000000011111111,D4   ; SELECT RED
        LSL.L   #8,D4                   ; <- 8 bits (so from 8 to 15)
        LSL.L   #8,D4                   ; <- 8 bits (so from 0 to 7)
        OR.L    D4,D5                   ; SAVE RED TO D5
        MOVE.L  D5,(A1)+                ; SAVE 24 BIT VALUE IN TAB
        DBRA    D7,COLCALCLOOP          ; 256 TIMES FOR 256 COLORS

        ADDQ.W  #1,D6           ; ADD 1 TO MULTIPLIER
        CMPI.W  #255,D6         ; MULTIPLIER MAX = 256
        BLE.S   FADESTEPS       ; IF NOT MAX NEXT FADE STEP
        RTS


*****************************************************************************
* Questa routine converte i colori a 24 bit, che si presentano come una      *
* longword $00RrGgBb, (dove R = nibble alto di RED, r = nibble basso di RED, *
* G = nibble alto di GREEN eccetera), nel formato della copperlist aga,      *
* ossia in due word: $0RGB con i nibble alti e $0rgb con i nibble bassi.     *
*****************************************************************************

MettiColori:
        MOVE.L  ActualFadeTab(PC),A0    ; indirizzo della color palette al
                                        ; punto attuale del fade dalla TAB
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
```

```
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.


;*****************************************************************************
;*                            COPPERLIST AGA                                 *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

                ; 5432109876543210
        dc.w    $100,%0000001000010001  ; 8 bitplane LOWRES 320x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,0          ; Burst mode azzerato (per ora!)

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
```

```
        dc.w $f4,0,$f6,0        ; sesto           "
        dc.w $f8,0,$fA,0        ; settimo         "
        dc.w $fC,0,$fE,0        ; ottavo          "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0
```

```
        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori
```

```
        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

*****************************************************************************

        Section BufPerPrecalc,BSS       ; va benissimo anche in fast!

; 256 COLORI.L * 256

COLORTABBY:
        DS.B    4*256*256       ; 262144 bytes da precalcolare!
CTABEND:

        end
```

In fondo e' un "upgrade" della vecchia routine di fade! No?

Lezione15c4

```
; Lezione15c4.s         - Seconda prova di fade a 24 bit. Ora la tabella
;                         contiene i colori gia' "convertiti" in nibble
;                         bassi e alti per metterli in coplist.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num of bitplanes -1
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #40*256,d0      ; lenght of bitplane
        addq.w  #8,a1
        dbra    d7,POINTB       ; Rifai D7 volte (D7=num of bitplanes)

        bsr.w   FADE256PRECALC  ; Precalcola i valori di tutto il fade, per
                                ; un totale di 256 colori.l in 256 passaggi dal
                                ; nero al colore pieno, ossia 4*256*256 bytes
                                ; di tabella: 262144 bytes precalcolati!!!
```

```
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        bsr.s   MainFadeInOut   ; Routine che sfuma dal nero al colore pieno
                                ; e viceversa.

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS


*****************************************************************************
* Questa routine fa scattare in avanti o indietro il puntatore ai colori    *
* precalcolati ActualFadeTab. Quando il fade va dal nero al colore pieno    *
* aggiunge 256 longwords al puntatore, facendolo puntare ai prossimi 256    *
* colori, ossia al prossimo fotogramma piu' scuro precalcolato. Nel caso    *
* opposto torna indietro al fotogramma precedente. La label FlagFadeInOut   *
* e' usata per controllare se il fade e' In o Out.                          *
*****************************************************************************

MainFadeInOut:
        BSR.w   MettiColori2    ; Sistema i colori di questo fotogramma
                                ; prendendoli dalla tabella (gia' convertiti)
        BTST.b  #1,FlagFadeInOut ; Fade In o fade Out?
        BNE.S   FadeOut
FadeIn:
        ADD.L   #256*4,ActualFadeTab    ; Prossimo fotogramma (256 colori .l)
        LEA     CTABEND,A0              ; Indirizzo fine tabella
        CMP.L   ActualFadeTab(PC),A0    ; Siamo attivati alla fine della tab
                                        ; del fade? (Colori pieni e lucenti)
        BNE.s   NonFinito
        BCHG.B  #1,FlagFadeInOut        ; Cambia la direzione del fade
FadeOut:
        SUB.L   #256*4,ActualFadeTab    ; Passo precedente (piu' scuro)
        LEA     COLORTABBY,A0           ; Indirizzo di inizio tabella
        CMP.L   ActualFadeTab(PC),A0    ; Siamo arrivati all'inizio della tab
                                        ; del fade? (Colore NERO)
        BNE.W   NonFinito
        BCHG.B  #1,FlagFadeInOut        ; cambia la direzione del fade
NonFinito:
        RTS
```

```
FlagFadeInOut:          ; Usato per decidere se FadeIn o FadeOut
        dc.w    0

ActualFadeTab:                  ; Puntatore al "fotogramma" precalcolato del
        dc.l    COLORTABBY      ; fade nella tabella COLORTABBY.


*****************************************************************************
* Questa routine precalcola tutti i colori a 24 bit del fade, e fa un bel po'*
* di lavoro, dato che deve scrivere 256*256 longwords, ossia 262144 bytes!   *
* Non e' altro che la routine di fade usata per i colori a 12 bit dell'amiga *
* normale, solo che tratta 1 byte per componente RGB, anziche' 4 bit.        *
* E' inoltre fatta la conversione in words per la copperlist AGA.            *
*****************************************************************************

FADE256PRECALC:
        LEA     COLORTABBY,A1   ; DEST CALCULATED COLORS TAB
        LEA     temporaneo(PC),A2 ; DEST CALCULATED COLORS TAB
        MOVEQ   #0,D6           ; MULTIPLIER START (0-255)
FADESTEPS:
        LEA     PICTURE+(10240*8),A0    ; 24bit colors tab address
        MOVE.w  #256-1,D7               ; NUM. DI COLORI = 256

COLCALCLOOP:

;       CALCOLA IL BLU

        MOVE.L  (A0),D4                 ; READ COLOR FROM TAB
        ANDI.L  #%000011111111,D4       ; SELECT BLUE
        MULU.W  D6,D4                   ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%000011111111,D4       ; SELECT BLUE VAL
        MOVE.L  D4,D5                   ; SAVE BLUE TO D5

;       CALCOLA IL VERDE

        MOVE.L  (A0),D4                 ; READ COLOR FROM TAB
        ANDI.L  #%1111111100000000,D4   ; SELECT GREEN
        LSR.L   #8,D4                   ; -> 8 bits (so from 0 to 7)
        MULU.W  D6,D4                   ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%0000000011111111,D4   ; SELECT GREEN
        LSL.L   #8,D4                   ; <- 8 bits (so from 8 to 15)
        OR.L    D4,D5                   ; SAVE GREEN TO D5

;       CALCOLA IL BLU

        MOVE.L  (A0)+,D4                ; READ COLOR FROM TAB AND GO TO NEXT
        ANDI.L  #%111111110000000000000000,D4   ; SELECT RED
        LSR.L   #8,D4                   ; -> 8 bits (so from 8 to 15)
        LSR.L   #8,D4                   ; -> 8 bits (so from 0 to 7)
        MULU.W  D6,D4                   ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%0000000011111111,D4   ; SELECT RED
        LSL.L   #8,D4                   ; <- 8 bits (so from 8 to 15)
        LSL.L   #8,D4                   ; <- 8 bits (so from 0 to 7)
        OR.L    D4,D5                   ; SAVE RED TO D5
        MOVE.L  D5,(A2)                 ; SAVE 24 BIT VALUE IN temporaneo

;***
```

```
; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A2),(a1)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a1) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a2),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a2),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a1)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A2),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,2(a1)        ; Copia il byte alto $0R nel color register
        move.b  2(a2),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a2),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1+2(a1)      ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
;****
        DBRA    D7,COLCALCLOOP          ; 256 TIMES FOR 256 COLORS

        ADDQ.W  #1,D6           ; ADD 1 TO MULTIPLIER
        CMPI.W  #255,D6         ; MULTIPLIER MAX = 256
        BLE.W   FADESTEPS       ; IF NOT MAX NEXT FADE STEP
        RTS

Temporaneo:
        dc.l    0

*****************************************************************************
* Questa routine converte i colori a 24 bit, che si presentano come una      *
* longword $00RrGgBb, (dove R = nibble alto di RED, r = nibble basso di RED, *
* G = nibble alto di GREEN eccetera), nel formato della copperlist aga,      *
* ossia in due word: $0RGB con i nibble alti e $0rgb con i nibble bassi.     *
*****************************************************************************


MettiColori2:
        MOVE.L  ActualFadeTab(PC),A0    ; indirizzo della color palette al
                                        ; punto attuale del fade dalla TAB
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
```

```
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
Fai8Banchi:
        moveq   #32-1,d6        ; 32 registri colore per banco

MettiBank:      ; loop che mette un banco da 32 registri

        move.l  (a0)+,d0        ; copia le 2 word $0rgb0RGB in d0

; nibble alti

        move.w  d0,(a1)         ; copia $0RGB

; nibble bassi

        swap    d0
        move.w  d0,(a2)         ; copia $0rgb

        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,MettiBank

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,Fai8Banchi   ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.


;***************************************************************************
;*                              COPPERLIST AGA                             *
;***************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

                ; 5432109876543210
        dc.w    $100,%0000001000010001  ; 8 bitplane LOWRES 320x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,0          ; Burst mode azzerato (per ora!)

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
```

```
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI
```

```
DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
;*****************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

*****************************************************************************

        Section BufPerPrecalc,BSS       ; va benissimo anche in fast!

; 256 COLORI.L * 256

COLORTABBY:
        DS.B    4*256*256       ; 262144 bytes da precalcolare!
CTABEND:

        end
```

Questa volta nella tabella COLORTABBY sono salvate direttamente le words per
i registri, dato che viene precalcolata anche la conversione da:

```
        $00RrGgBb       a       $0rgb0RGB
```

Ossia dalla long con il colore a 24 bit alla coppia di registri word.
In questo modo la routine che "sfuma" deve solo copiare le word, senza la
conversione, ed e' piu' veloce: si possono compiere piu' operazioni mentre
avviene il fade.

Lezione15c5

```
; Lezione15c5.s -       Fade a 24bit in tempo reale, non precalcolato

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num of bitplanes -1
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
```

```
        swap    d0
        addi.l  #10240,d0       ; lenght of bitplane
        addq.w  #8,a1
        dbra    d7,POINTB       ; Rifai D7 volte (D7=num of bitplanes)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$08000,d2      ; linea da aspettare = $80
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $80
        BNE.S   Waity1

        bsr.s   MainFadeInOut   ; Routine che sfuma dal nero al colore pieno
                                ; e viceversa.

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$08000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

*****************************************************************************
* Questa routine incrementa o decrementa il MULTIPLIER per il fadeIn/Out    *
* FlagFadeInOut e' usata per controllare se il fade e' In o Out.            *
*****************************************************************************

MainFadeInOut:
        BSR.w   CalcolaMettiCol ; Calcola i 256 colori in questa fase del
                                ; fade, a seconda del MULTIPLIER, li converte
                                ; nelle word per la copperlist AGA e li copia
                                ; nella copperlist stessa.

        BTST.b  #1,FlagFadeInOut ; Fade In o fade Out?
        BNE.S   FadeOut
FadeIn:
        ADDQ.W  #1,MULTIPLIER   ; Prossima fase del fade (piu' chiaro)
        CMP.W   #255,MULTIPLIER ; Siamo arrivati alla massima chiarezza
                                ; del fade? (Colori pieni e lucenti)
        BNE.s   NonFinito       ; Se non ancora, -> NonFinito
        BCHG.B  #1,FlagFadeInOut ; Altrimenti cambia la direzione del fade
FadeOut:
        SUBQ.W  #1,MULTIPLIER   ; Prossima fase del fade (piu' scuro)
        BNE.W   NonFinito       ; multiplier=zero? Se non ancora -> Nonfinito
        BCHG.B  #1,FlagFadeInOut ; Altrimenti cambia la direzione del fade
NonFinito:
        RTS
```

```
FlagFadeInOut:          ; Usato per decidere se FadeIn o FadeOut
        dc.w    0

MULTIPLIER:
        dc.w    0

Temporaneo:
        dc.l    0

*****************************************************************************
* Questa routine converte i colori a 24 bit, che si presentano come una      *
* longword $00RrGgBb, (dove R = nibble alto di RED, r = nibble basso di RED, *
* G = nibble alto di GREEN eccetera), nel formato della copperlist aga,      *
* ossia in due word: $0RGB con i nibble alti e $0rgb con i nibble bassi.     *
*****************************************************************************

CalcolaMettiCol:
        LEA     temporaneo(PC),A0       ; Long temporanea per colore a 24
                                        ; bit nel formato $00RrGgBb
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        LEA     palettepic(PC),A3       ; 24bit colors tab address

        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

;       CALCOLA IL ROSSO

        MOVE.L  (A3),D4                 ; READ COLOR FROM TAB
        ANDI.L  #%000011111111,D4       ; SELECT BLUE
        MULU.W  MULTIPLIER(PC),D4               ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%000011111111,D4       ; SELECT BLUE VAL
        MOVE.L  D4,D5                   ; SAVE BLUE TO D5

;       CALCOLA IL VERDE

        MOVE.L  (A3),D4                 ; READ COLOR FROM TAB
        ANDI.L  #%1111111100000000,D4   ; SELECT GREEN
        LSR.L   #8,D4                   ; -> 8 bits (so from 0 to 7)
        MULU.W  MULTIPLIER(PC),D4       ; MULTIPLIER
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%0000000011111111,D4   ; SELECT GREEN
        LSL.L   #8,D4                   ; <- 8 bits (so from 8 to 15)
        OR.L    D4,D5                   ; SAVE GREEN TO D5

;       CALCOLA IL BLU

        MOVE.L  (A3)+,D4                ; READ COLOR FROM TAB AND GO TO NEXT
        ANDI.L  #%111111110000000000000000,D4   ; SELECT RED
        LSR.L   #8,D4                   ; -> 8 bits (so from 8 to 15)
        LSR.L   #8,D4                   ; -> 8 bits (so from 0 to 7)
        MULU.W  MULTIPLIER(PC),D4       ; MULTIPLIER
```

```
        ASR.w   #8,D4                   ; -> 8 BITS
        ANDI.L  #%0000000011111111,D4   ; SELECT RED
        LSL.L   #8,D4                   ; <- 8 bits (so from 8 to 15)
        LSL.L   #8,D4                   ; <- 8 bits (so from 0 to 7)
        OR.L    D4,D5                   ; SAVE RED TO D5
        MOVE.L  D5,(A0)                 ; SAVE 24 BIT VALUE IN temporaneo

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Tabella con la palette a 24 bit in formato $00RRGGBB. Avremmo potuto anche
; usare quella attaccata in fondo alla PIC, ma per variare eccola in dc.l!
; Si puo' salvare da PicCon se non si seleziona "Copperlist".

PalettePic:
        dc.l    $021104,$150f04,$001115,$191609,$092206,$182707
```

```
        dc.l    $052420,$2f1506,$17291c,$1f3108,$341613,$35230b
        dc.l    $1c331c,$2c3409,$00203d,$35241f,$323420,$21470a
        dc.l    $103937,$4a2007,$47201b,$243a32,$002a4a,$35440d
        dc.l    $492822,$443c0a,$21550a,$54280b,$483421,$3a3931
        dc.l    $07364f,$233e45,$1d4d3c,$32590c,$01335d,$27503d
        dc.l    $484f11,$5a3e0e,$354e3c,$5c3921,$593431,$70230e
        dc.l    $4c5b12,$064066,$5b442c,$5d4d11,$465a30,$104367
        dc.l    $732e11,$316143,$5b4838,$324662,$506714,$763b0f
        dc.l    $704023,$655711,$4d5d44,$733f31,$19536e,$8a2012
        dc.l    $2b6261,$6c552e,$784f19,$0d4c7d,$79492e,$8f2713
        dc.l    $716217,$6c612b,$455e61,$8f370f,$1b5081,$705845
        dc.l    $716e16,$943c17,$516463,$8b4e1f,$1e5f83,$8f510f
        dc.l    $746d2d,$89512e,$588a24,$776a45,$8c631d,$8e5e2f
        dc.l    $72635d,$8c5644,$b02015,$1d6791,$aa3c15,$af2d14
        dc.l    $8d7419,$2b6d90,$a1552d,$788a2a,$a25f13,$936d31
        dc.l    $ac4e13,$6e7071,$3a7192,$bd2a16,$878b1e,$a1672e
        dc.l    $926f47,$a46e13,$5d8e67,$8d7054,$a06546,$2f739f
        dc.l    $a37331,$c92b16,$b66110,$8e8b3b,$818d55,$c74013
        dc.l    $3d79a4,$8c7173,$b36e2b,$ba6c11,$ad7244,$a77253
        dc.l    $a58b27,$8f8c60,$a58444,$4489a4,$a59720,$a77661
        dc.l    $cd6214,$ba763e,$a78f41,$db4114,$a48b55,$4589b0
        dc.l    $b87754,$608fa2,$c97728,$8ba268,$d46d10,$c58428
        dc.l    $b8894a,$c88614,$a48c6e,$d86b22,$a59e59,$898f97
        dc.l    $5a94b3,$e46217,$c59427,$c98940,$b99259,$df6a39
        dc.l    $c39443,$c2a025,$a79a74,$da734c,$5595c1,$8c9f95
        dc.l    $c79156,$b99271,$ef6327,$ea7515,$de8b1a,$a89988
        dc.l    $eb7623,$df8d29,$c7a93e,$dd903f,$669ac7,$c5a55b
        dc.l    $c79770,$5da6c8,$f08a18,$d6995f,$ea971a,$dda043
        dc.l    $f3872b,$e1a42d,$caa472,$a8a2a3,$71a7cb,$ef9341
        dc.l    $80a6c6,$caae6f,$f2982a,$c7a287,$f69a1c,$e99959
        dc.l    $d7a86a,$f2a13e,$ccb678,$c6a499,$efb127,$e8b247
        dc.l    $89b2c9,$e5a868,$c8af94,$e2b363,$bcc28f,$f7af2d
        dc.l    $deb577,$8bb1d6,$d1b295,$beb5aa,$f7b34b,$dbb191
        dc.l    $f5b462,$8cb3e3,$f1b375,$d2ba9f,$fdc42f,$dfc189
        dc.l    $fac34a,$87c0e3,$c3c4ae,$a6bbd4,$e2ba94,$fbd232
        dc.l    $f3c967,$98bee4,$f0bf85,$d6c6a4,$fbd24a,$d3baba
        dc.l    $c1bfcd,$e8ce8d,$fdd457,$a3c3e9,$dbd0a8,$d3cbbd
        dc.l    $f0c4a0,$fcdb66,$adcde0,$f7cf8a,$b3c5e9,$dfcebc
        dc.l    $c4d2d8,$feea66,$f3e489,$b5ceeb,$d2ced4,$f5d0a8
        dc.l    $fee885,$f3dfa7,$d8dfca,$c5d2ec,$f5d2bc,$d1d2e7
        dc.l    $ede1c2,$d9d4e7,$fde7aa,$f2d2d5,$f9e2c0,$d2dfef
        dc.l    $e9dedf,$f9f3c4,$f8efda,$f9f5ee

;***************************************************************************
;*                              COPPERLIST AGA                             *
;***************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod

                    ; 5432109876543210
```

```
        dc.w    $100,%0000001000010001  ; 8 bitplane LOWRES 320x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,0          ; Burst mode azzerato (per ora!)

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0
```

```
DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
```

```
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

        end
```

Abbiamo eliminato la COLORTABBY, e questa si puo' chiamare "FADE IN REALTIME", dato che viene calcolato fotogramma per fotogramma. E' molto piu' lenta di quelli precalcolati, ma non richiede 256k di buffer. Si puo' usare quando occorre fare il fade di una figura statica o comunque dove non ci sono routines molto "mangiatempo". Da notare che la palette viene presa da una tabella, anziche' dalla fine della pic.

29.4 Lezione15d

```
; Lezione15d.s  - Visualiziamo la prima pic in hires a 256 colori con
;                 il burst pompato a 64 bit (FMODE, $1fc=3)

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #80*100,d0      ; Lunghezza di un bitplane
```

```
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
        move.w  #99-1,d7        ; Numero linee: 99
        bsr.w   FaiAGACopB      ; Fai una sfumatura BLU

        bsr.s   MettiColori

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS


; Questa routine, che e' presente anche nella mia demo WORLD OF MANGA, serve
; per leggere la palette a 24 bit, in questo caso inclusa con un INCBIN
; In pratica converte ogni colore a 24 bit, che si presenta nel formato di
; una long $00RrGgBb, dove R = nibble alto di RED, r = nibble basso di RED,
; G = nibble alto di GREEN eccetera, nel formato della copperlist aga, ossia
; in due word $0RGB con i nibble alti e $0rgb con i nibble bassi.

MettiColori:
        LEA     LogoPal(PC),A0          ; indirizzo della color palette
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi
```

```
; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Palette salvata in binario con il PicCon (opzioni: save as binary, non cop)

LogoPal:
        incbin  "Pic640x100x256.pal"

;******************************************************************************
; Routine che crea sfumature aga BLU:
;
; d1 = Prima linea da aspettare (Wait, ad es: $2c07fffe per linea Y=$2c)
; d5 = inizio tonalita' ($00-$ff)
; d7 = Numero linee da fare
;******************************************************************************

FaiAgaCopB:
        lea     AgaCopEff1,a0
        move.l  #$01060c00,d4   ; BplCon3 - nibble alti
        move.l  #$01060e00,d3   ; BplCon3 - nibble bassi
        move.w  #$180,d2        ; Registro Color0
FaiAGALoopB:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        move.l  d4,(a0)+        ; BplCon3 - selez. nibble alti
        move.w  d2,(a0)+        ; Registro Color0
        addq.b  #1,d5           ; "Illumina" leggermente il colore $Gg
        move.w  d5,d6           ; Copialo in d6
        and.w   #%11110000,d6   ; Selez. solo il nibble ALTO
        lsr.w   #4,d6           ; Alla posizione giusta, ossia al BLU $xxB)
```

```
        move.w  d6,(a0)+        ; Valore del Color0 (nib alti)
        move.l  d3,(a0)+        ; BplCon3 - selez. nibble bassi
        move.w  d2,(a0)+        ; Registro Color0
        move.w  d5,d6           ; Colore $xx in d6
        and.w   #%00001111,d6   ; Selez. solo i nibble bassi - posizione $xxB
        move.w  d6,(a0)+        ; Metti il colore in copperlist (nibble bassi)
        dbra    d7,FaiAGALoopB
        rts

;*****************************************************************************
;*                            COPPERLIST                                     *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

; Nota: il ddfstart/stop HIRES sarebbero $003c e $00d4, ma con il burst attivo
; va bene lo stesso valore del LOWRES, ossia $0038 e $00d0.

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%1000001000010001  ; 8 bitplane HIRES 640x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,3          ; Burst mode a 64 bit

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
```

```
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI
```

```
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

AgaCopEff1:
        dcb.l   99*5            ; Ossia: 99 linee * 5 long:
                                ; 1 per il wait,
                                ; 1 per il bplcon3
                                ; 1 per color0 (nib alti)
                                ; 1 per il bplcon3
                                ; 1 per color0 (nib bassi)
        dc.w    $9007,$fffe     ; aspetta la fine del logo
        dc.w    $100,$200       ; zero bitplanes

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "Pic640x100x256.RAW"    ; (C) by Cristiano "KREEX" Evangelisti
```

```
        end

Da notare, nella copperlist, che i moduli sono -8, perche' l'FMODE ($1fc) e'
a 64 bit. Provate a togliere il CNOP 0,8 davanti alla label PICTURE, con un
bel punto e virgola, e noterete che assemblando la pic ad un indirizzo non
allineato a 64 bit otterrete una visualizzazione a fette verticali.
Altro particolare e' che con il burst attivo, il DDFSTART e il DDFSTOP non
vanno messi al valore per l'hires normale, ossia $003c e $00d4, ma allo stesso
valore del lowres, dato che "pompa" velocemente.

        dc.w    $92,$0038       ; DdfStart lowres, adatto per HIRES con burst
        dc.w    $94,$00d0       ; DdfStop lowres, come sopra

Ci sta bene la sfumatura copper blu sullo sfondo, vero? Almeno andiamo a
256+99 colori totali visualizzati, ossia 355.
```

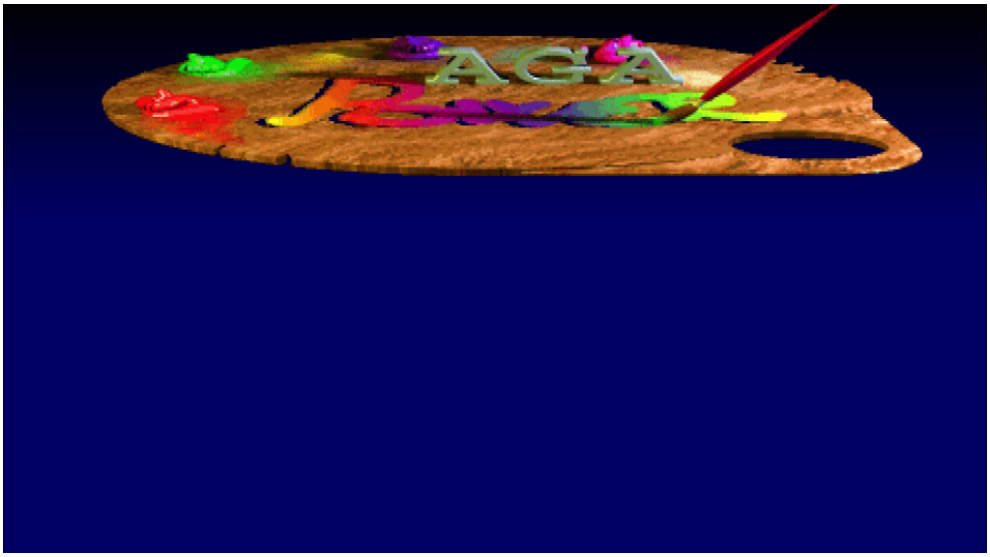


Figura 29.3: Lezione 15d

29.5 Lezione15e

```
; Lezione15e.s  - Visualiziamo la prima pic in ham8 a 256000 possibili colori.
;                 DA NOTARE CHE BASTA SETTARE 64 REGISTRI COLORE PER LA
;                 PALETTE, I RESTANTI 192 SONO IGNORATI.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
```

```
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #80*100,d0      ; Lunghezza di un bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        bsr.s   MettiColori

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        BTST    #6,$BFE001      ; aspetta il mouse
        BNE.S   LOOP
        RTS


; Questa routine, che e' presente anche nella mia demo WORLD OF MANGA, serve
; per leggere la palette a 24 bit, in questo caso inclusa con un INCBIN
; In pratica converte ogni colore a 24 bit, che si presenta nel formato di
; una long $00RrGgBb, dove R = nibble alto di RED, r = nibble basso di RED,
; G = nibble alto di GREEN eccetera, nel formato della copperlist aga, ossia
; in due word $0RGB con i nibble alti e $0rgb con i nibble bassi.

MettiColori:
        LEA     LogoPal(PC),A0          ; indirizzo della color palette
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #2-1,d7                 ; 2 banchi da 32 registri ciascuno
                                        ; *NOTA: IN HAM8 BASTA DEFINIRE UNA
                                        ; PALETTE DI 64 COLORI! GLI ALTRI
                                        ; REGISTRI COLORE DAL 65 AL 255 SONO
                                        ; IGNORATI TOTALMENTE!
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.
```

```
; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Palette salvata in binario con il PicCon (opzioni: save as binary, non cop)
; NOTA: ESSENDO LA PALETTE DI UNA PIC HAM, CONTIENE SOLO 64 COLORI, NON 256.

LogoPal:
        incbin  "pic640x100xham8.pal"

;******************************************************************************
;*                              COPPERLIST                                    *
;******************************************************************************

        CNOP    0,8     ; Allineo a 64 bit
```

```
        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

; Nota: il ddfstart/stop HIRES sarebbero $003c e $00d4, ma con il burst attivo
; va bene lo stesso valore del LOWRES, ossia $0038 e $00d0.

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%1000101000010001  ; 8 bitplane HIRES 640x256 HAM8. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.
                                        ; Settando il bit 11 attivo l'HAM8

        dc.w    $1fc,3          ; Burst mode a 64 bit

; Il RAW e' salvato con PicCon, dunque si puo' puntare normalmente.

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Questo e' l'ordine dei bitplanes se salvate il RAW con AgaConv
;
;       dc.w $e8,0,$ea,0        ; terzo        bitplane
;       dc.w $ec,0,$ee,0        ; quarto           "
;       dc.w $f0,0,$f2,0        ; quinto           "
;       dc.w $f4,0,$f6,0        ; sesto            "
;       dc.w $f8,0,$fA,0        ; settimo          "
;       dc.w $fC,0,$fE,0        ; ottavo           "
;       dc.w $e0,0,$e2,0        ; primo            "
;       dc.w $e4,0,$e6,0        ; secondo          "


; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

; *NOTA: IN HAM8 BASTA DEFINIRE 64 COLORI, NON TUTTI E 255!!!!!

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0
```

```
        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLPOB:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $9007,$fffe     ; aspetta la fine del logo
        dc.w    $100,$200       ; zero bitplanes

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;***************************************************************************

; Figura RAW ad 8 bitplanes, in HAM8.

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "pic640x100xham8.RAW"

        end
```

La particolarta' che avrete notato, e' che la palette e' composta da solo 64 colori. Basta settare il bit 11 di bplcon0, quello dell'HAM, e il gioco e' fatto. Non dimenticatevi il particolare dello scambio dei bitplanes (puntandoli come nell'esempio in copperlist) nel caso che salviate il RAW con un Iffconverter che non li scambia da solo, come invece fa il PicCon.

Lezione15e2

```
; Lezione15e2.s - Visualiziamo due pic hires insieme, una a 256 colori e
;                 l'altra in HAM8. Notate differenza?

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA
```

```
WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #80*100,d0      ; Lunghezza di un bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        MOVE.L  #PICTUREham,d0
        LEA     BPLPOINTERSham,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
POINTBham:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #80*100,d0      ; Lunghezza di un bitplane
        addq.w  #8,a1
        dbra    d7,POINTBham            ;Rifai D1 volte (D1=num of bitplanes)


; Per la figura 256 colori

        LEA     LogoPal(PC),A0          ; indirizzo della color palette
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
        bsr.s   MettiColori


; Per la figura HAM8

        LEA     LogoPalham(PC),A0       ; indirizzo della color palette
        LEA     COLP0ham+2,A1           ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0Bham+2,A2          ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #2-1,d7                 ; 2 banchi da 32 registri ciascuno
        bsr.s   MettiColori


        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        BTST    #6,$BFE001
```

```
        BNE.S   LOOP
        RTS


; Questa routine, che e' presente anche nella mia demo WORLD OF MANGA, serve
; per leggere la palette a 24 bit, in questo caso inclusa con un INCBIN
; In pratica converte ogni colore a 24 bit, che si presenta nel formato di
; una long $00RrGgBb, dove R = nibble alto di RED, r = nibble basso di RED,
; G = nibble alto di GREEN eccetera, nel formato della copperlist aga, ossia
; in due word $0RGB con i nibble alti e $0rgb con i nibble bassi.

MettiColori:
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist
```

```
        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Palette salvata in binario con il PicCon (opzioni: save as binary, non cop)

LogoPal:
        incbin  "Pic640x100x256.pal"

LogoPalHAM:
        incbin  "pic640x100xham8.pal"

;******************************************************************************
;*                              COPPERLIST                                    *
;******************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

; Nota: il ddfstart/stop HIRES sarebbero $003c e $00d4, ma con il burst attivo
; va bene lo stesso valore del LOWRES, ossia $0038 e $00d0.

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%1000001000010001  ; 8 bitplane HIRES 640x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,3          ; Burst mode a 64 bit

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.
```

```
        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0
```

```
        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $9007,$fffe     ; aspetta la fine del logo
        dc.w    $100,$201       ; zero bitplanes

; Il RAW e' salvato con PicCon, dunque si puo' puntare normalmente.

BPLPOINTERSham:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
```

```
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Questo e' l'ordine dei bitplanes se salvate il RAW con AgaConv
;
;       dc.w $e8,0,$ea,0        ; terzo        bitplane
;       dc.w $ec,0,$ee,0        ; quarto           "
;       dc.w $f0,0,$f2,0        ; quinto           "
;       dc.w $f4,0,$f6,0        ; sesto            "
;       dc.w $f8,0,$fA,0        ; settimo          "
;       dc.w $fC,0,$fE,0        ; ottavo           "
;       dc.w $e0,0,$e2,0        ; primo            "
;       dc.w $e4,0,$e6,0        ; secondo          "


; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

; *NOTA: IN HAM8 BASTA DEFINIRE 64 COLORI, NON TUTTI E 255!!!!!

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0ham:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0Bham:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $9507,$fffe

                ; 5432109876543210
        dc.w    $100,%1000101000010001  ; 8 bitplane HIRES 640x256 HAM8. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.
                                        ; Settando il bit 11 attivo l'HAM8

        dc.w    $f907,$fffe
        dc.w    $100,$200

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
;*****************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "Pic640x100x256.RAW"    ; (C) by Cristiano "KREEX" Evangelisti

; Figura RAW ad 8 bitplanes, in HAM8.

        CNOP    0,8     ; allineo a 64 bit

PICTUREHAM:
        INCBIN  "pic640x100xham8.RAW"

        end

Sembrano uguali! Eppure sopra sono 256 colori, sotto ham8!
Forse nel power si nota qualche pixel nella versione a 256 colori...
Comunque l'ham8 si noterebbe meglio con una foto scannerizzata.
```

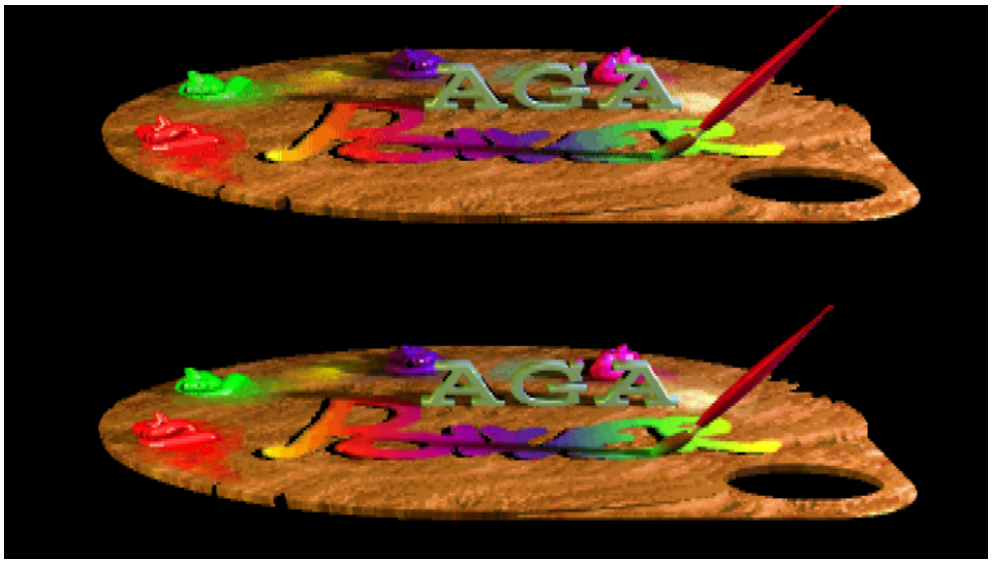


Figura 29.4: Lezione 15e2

29.6 Lezione15f

```
; Lezione15f.s          Sprite in HIRES, largo 16 pixel. Usare il tasto destro
;                       del mouse per scambiare tra LowRes e HighRes

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
        dbra    d1,NulLoop

;       Puntiamo lo sprite

        MOVE.L  #MIOSPRITE,d0           ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

        move.b  $dff00a,mouse_y
        move.b  $dff00b,mouse_x

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        bsr.s   LeggiMouse      ; questa legge il mouse
        move.w  sprite_y(pc),d0 ; prepara i parametri per la routine
```

```
        move.w  sprite_x(pc),d1 ; universale
        lea     miosprite,a1    ; indirizzo sprite
        moveq   #13,d2          ; altezza sprite
        bsr.w   UniMuoviSprite  ; chiama la routine universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst.b  #2,$dff016      ; Tasto destro premuto?
        bne.s   NonScambiareRes ; Se no non cambiare risuluzione allo sprite

        bchg.b  #7,BplCon3      ; Se si, cambia da LowRes a Hires o viceversa.

NonScambiareRes:
        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b  $dff00a,d1      ; JOYODAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_y     ; modifica posizione sprite
no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_x     ; modifica pos. sprite
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

SPRITE_Y:       dc.w    0       ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.w    0       ; qui viene memorizzata la X dello sprite
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse


; Routine universale di posizionamento degli sprite.

;
;       Parametri in entrata di UniMuoviSprite:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
```

```
;       d2 = altezza dello sprite
;

UniMuoviSprite:
; posizionamento verticale
        ADD.W   #$2c,d0         ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b  d0,(a1)         ; copia il byte in VSTART
        btst.l  #8,d0
        beq.s   NonVSTARTSET
        bset.b  #2,3(a1)        ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,3(a1)        ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   D2,D0           ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,2(a1)        ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,3(a1)        ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,3(a1)        ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w   #128,D1         ; 128 - per centrare lo sprite.
        btst    #0,D1           ; bit basso della coordinata X azzerato?
        beq.s   BitBassoZERO
        bset    #0,3(a1)        ; Settiamo il bit basso di HSTART
        bra.s   PlaceCoords

BitBassoZERO:
        bclr    #0,3(a1)        ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w   #1,D1           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b  D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
```

```
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,3          ; Burst mode a 64 bit

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w    $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w    $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO


        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$000       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$462       ; color17, nibble bassi
        dc.w    $1A4,$2e4       ; color18, nibble bassi
        dc.w    $1A6,$672       ; color19, nibble bassi

        dc.w    $106    ; BPLCON3
        dc.b    0
BplCon3:
               ; 76543210
        dc.b    %10000000       ; bit 7: sprites hires o lowres. Se si
                                ; settano sia il bit 7 che il bit 6 lo sprite
                                ; e' in superhires (1280x256), ma viene troppo
                                ; "stretto", credo sia inutile, basta hires!

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;*****************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;*****************************************************************************
```

```
        cnop    0,8

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
        dc.l    0,0,0,0         ; negli eventuali puntatori inutilizzati


        cnop    0,8

MIOSPRITE:              ; lunghezza 13 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit

 dc.w   %0000000000000000,%0000110000110000 ; Formato binario per modifiche
 dc.w   %0000000000000000,%0000011001100000
 dc.w   %0000000000000000,%0000001001000000
 dc.w   %0000000110000000,%0011000110001100 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 dc.w   %0000011111100000,%0110011111100110 ;BINARIO 10=COLORE 1 (ROSSO)
 dc.w   %0000011111100000,%1100100110010011 ;BINARIO 01=COLORE 2 (VERDE)
 dc.w   %0000110110110000,%1111100110011111 ;BINARIO 11=COLORE 3 (GIALLO)
 dc.w   %0000011111100000,%0000011111100000
 dc.w   %0000011111100000,%0001111001111000
 dc.w   %0000001111000000,%0011101111011100
 dc.w   %0000000110000000,%0011000110001100
 dc.w   %0000000000000000,%1111000000001111
 dc.w   %0000000000000000,%1111000000001111
 dc.w   0,0     ; 2 word azzerate definiscono la fine dello sprite.


        cnop    0,8

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

In questo listato si notano gia' 2 cose sugli sprite AGA: una cosa e' come selezionarne la risoluzione, tra LowRes, Hires o SuperHires. In realta' la risoluzione SuperHires e' inutile, perche' lo sprite viene troppo piccolo. Un'altra cosa e' che la palette dello sprite e' a 24 bit, come i bitplanes, per cui si mettono prima i nibble alti, poi i nibble bassi.

Lezione15f2

```
; Lezione15f2.s         Sprite largo 32 pixel. Usare il tasto destro
;                       del mouse per scambiare tra LowRes e HighRes

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
        dbra    d1,NulLoop

;       Puntiamo lo sprite

        MOVE.L  #MIOSPRITE32,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

        move.b  $dff00a,mouse_y
        move.b  $dff00b,mouse_x

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        bsr.s   LeggiMouse      ; questa legge il mouse
        move.w  sprite_y(pc),d0 ; prepara i parametri per la routine
```

```
        move.w  sprite_x(pc),d1 ; universale
        lea     miosprite32,a1  ; indirizzo sprite
        moveq   #26,d2          ; altezza sprite
        bsr.w   UniMuoviSprite32 ; chiama la routine universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst.b  #2,$dff016      ; Tasto destro premuto?
        bne.s   NonScambiareRes ; Se no non cambiare risuluzione allo sprite

        bchg.b  #7,BplCon3      ; Se si, cambia da LowRes a Hires o viceversa.

NonScambiareRes:
        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b  $dff00a,d1      ; JOY0DAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e' fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_y     ; modifica posizione sprite
no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e' fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_x     ; modifica pos. sprite
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

SPRITE_Y:       dc.w    0       ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.w    0       ; qui viene memorizzata la X dello sprite
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse


; Routine universale di posizionamento degli sprite larghi 32 pixel.

;
;       Parametri in entrata di UniMuoviSprite32:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
```

```
;       d2 = altezza dello sprite
;

UniMuoviSprite32:
; posizionamento verticale
        ADD.W   #$2c,d0         ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b  d0,(a1)         ; copia il byte in VSTART
        btst.l  #8,d0
        beq.s   NonVSTARTSET
        bset.b  #2,3+2(a1)      ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,3+2(a1)      ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   D2,D0           ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,2+2(a1)      ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,3+2(a1)      ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,3+2(a1)      ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w   #128,D1         ; 128 - per centrare lo sprite.
        btst.l  #0,D1           ; bit basso della coordinata X azzerato?
        beq.s   BitBassoZERO
        bset.b  #0,3+2(a1)      ; Settiamo il bit basso di HSTART
        bra.s   PlaceCoords

BitBassoZERO:
        bclr.b  #0,3+2(a1)      ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w   #1,D1           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b  D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
```

```
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,%0111      ; Burst mode a 64 bit, sprite larghi 32 pixel

; Quest'altro settaggio e' simile. Provatelo se per problemi di DMA non
; funzionasse quello sopra.

;       dc.w    $1fc,%1011      ; Burst mode a 64 bit, sprite larghi 32 pixel

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w    $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w    $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO


        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$000       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$462       ; color17, nibble bassi
        dc.w    $1A4,$2e4       ; color18, nibble bassi
        dc.w    $1A6,$672       ; color19, nibble bassi

        dc.w    $106    ; BPLCON3
        dc.b    0
BplCon3:
               ; 76543210
        dc.b    %00000000       ; bit 7: sprites hires o lowres. Se si
                                ; settano sia il bit 7 che il bit 6 lo sprite
                                ; e' in superhires (1280x256), ma viene troppo
                                ; "stretto", credo sia inutile, basta hires!

        dc.w    $FFFF,$FFFE     ; Fine della copperlist
```

```
;******************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;******************************************************************************

        cnop    0,8

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
        dc.l    0,0,0,0         ; negli eventuali puntatori inutilizzati


        cnop    0,8

MIOSPRITE32:            ; lunghezza 13*2 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        DC.W 0          ; Word "aggiunta" nello sprite largo 32 pixel
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        DC.W 0          ; Word "aggiunta" nello sprite largo 32 pixel

 dc.l   %00000000000000000000000000000000,%00000000011000000000011000000000
 dc.l   %00000000000000000000000000000000,%00000000011100000000111000000000
 dc.l   %00000000000000000000000000000000,%00000000001110000001110000000000
 dc.l   %00000000000000000000000000000000,%00000000000111000011100000000000
 dc.l   %00000000000000000000000000000000,%00000000000011000011000000000000
 dc.l   %00000000000000000000000000000000,%00000000000011100111000000000000
 dc.l   %00000000000000011110000000000000,%00000111000001111110000011100000
 dc.l   %00000000000001111111000000000000,%00001110000001111111000000111000
 dc.l   %00000000001111111111100000000000,%00011100000111111111110000011100
 dc.l   %00000000011111111111110000000000,%01110000011001111110011000001110
 dc.l   %00000000111111111111111000000000,%11110000110000111100001100001111
 dc.l   %00000001111001111001111100000000,%11111111110000111100001111111111
 dc.l   %00000001111001111001111100000000,%01111111111000111100011111111110
 dc.l   %00000000111111111111111000000000,%00000000011111111111111000000000
 dc.l   %00000000011111111111110000000000,%00000001111111111111111100000000
 dc.l   %00000000001111111111110000000000,%00000011111111000011111110000000
 dc.l   %00000000000111111111100000000000,%00000111110011111111001111100000
 dc.l   %00000000000011111111000000000000,%00001111100011111111000111110000
 dc.l   %00000000000001111111000000000000,%00001111000001111111000011110000
 dc.l   %00000000000000110000000000000000,%00001111000000011000000011110000
 dc.l   %00000000000000000000000000000000,%11111111000000000000000011111111
 dc.l   %00000000000000000000000000000000,%11111111000000000000000011111111
 dc.l   %00000000000000000000000000000000,%11111111000000000000000011111111
 dc.l   %00000000000000000000000000000000,%11111111000000000000000011111111
 dc.l   0,0             ; Fine dello sprite (2 longword anziche' 2 word).

; BINARIO 00=COLORE 0 (TRASPARENTE)
; BINARIO 10=COLORE 1 (ROSSO)
; BINARIO 01=COLORE 2 (VERDE)
; BINARIO 11=COLORE 3 (GIALLO)

        cnop    0,8

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
```

```
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

Avete visto che bell'insettone largo 32 pixel?
La routine unimuovisprite e' stata modificata in modo molto semplice.
Infatti i 2 byte VSTOP e VHBITS si sono spostati una word avanti.
Per cui e' bastato sostituire:

```
        2(a1) e 3(a1)
```

In:

```
        2+2(a1) e 3+2(a1)
```

Niente di piu' facile! (Vi immaginerete come fare UniMuoviSprite 64 pixel!)

Lezione15f3

```
; Lezione15f3.s         Sprite largo 64 pixel. Usare il tasto destro
;                       del mouse per scambiare tra LowRes e HighRes

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
```

```
        dbra    d1,NulLoop

;       Puntiamo lo sprite

        MOVE.L  #MIOSPRITE64,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

        move.b  $dff00a,mouse_y
        move.b  $dff00b,mouse_x

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        bsr.s   LeggiMouse      ; questa legge il mouse
        move.w  sprite_y(pc),d0 ; prepara i parametri per la routine
        move.w  sprite_x(pc),d1 ; universale
        lea     miosprite64,a1  ; indirizzo sprite
        moveq   #52,d2          ; altezza sprite
        bsr.w   UniMuoviSprite64 ; chiama la routine universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst.b  #2,$dff016      ; Tasto destro premuto?
        bne.s   NonScambiareRes ; Se no non cambiare risuluzione allo sprite

        bchg.b  #7,BplCon3      ; Se si, cambia da LowRes a Hires o viceversa.

NonScambiareRes:
        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b  $dff00a,d1      ; JOYODAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
```

```
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_y     ; modifica posizione sprite
no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e' fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_x     ; modifica pos. sprite
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

SPRITE_Y:       dc.w    0       ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.w    0       ; qui viene memorizzata la X dello sprite
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse


; Routine universale di posizionamento degli sprite larghi 64 pixel.

;
;       Parametri in entrata di UniMuoviSprite64:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-255)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;       d2 = altezza dello sprite
;

UniMuoviSprite64:
; posizionamento verticale
        ADD.W   #$2c,d0         ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b  d0,(a1)         ; copia il byte in VSTART
        btst.l  #8,d0
        beq.s   NonVSTARTSET
        bset.b  #2,3+4+2(a1)    ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,3+4+2(a1)    ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   D2,D0           ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,2+4+2(a1)    ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,3+4+2(a1)    ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,3+4+2(a1)    ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale
        add.w   #128,D1         ; 128 - per centrare lo sprite.
        btst.l  #0,D1           ; bit basso della coordinata X azzerato?
        beq.s   BitBassoZERO
```

```
        bset.b  #0,3+4+2(a1)    ; Settiamo il bit basso di HSTART
        bra.s   PlaceCoords

BitBassoZERO:
        bclr.b  #0,3+4+2(a1)    ; Azzeriamo il bit basso di HSTART
PlaceCoords:
        lsr.w   #1,D1           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; il bit basso.
        move.b  D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,%1111      ; Burst mode a 64 bit, sprite larghi 64 pixel


BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.
```

```
        dc.w    $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w    $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w    $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO


        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLPOB:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$000       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$462       ; color17, nibble bassi
        dc.w    $1A4,$2e4       ; color18, nibble bassi
        dc.w    $1A6,$672       ; color19, nibble bassi

        dc.w    $106    ; BPLCON3
        dc.b    0
BplCon3:
               ; 76543210
        dc.b    %00000000       ; bit 7: sprites hires o lowres. Se si
                                ; settano sia il bit 7 che il bit 6 lo sprite
                                ; e' in superhires (1280x256), ma viene troppo
                                ; "stretto", credo sia inutile, basta hires!

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;******************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;******************************************************************************

        cnop    0,8

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
        dc.l    0,0,0,0         ; negli eventuali puntatori inutilizzati


        cnop    0,8

MIOSPRITE64:            ; lunghezza 13*4 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)

        dc.l    $00000000,$00000000,$00003000,$000c0000 ; Salvato con PicCon
        dc.l    $00000000,$00000000,$00003800,$001c0000
        dc.l    $00000000,$00000000,$00001c00,$00380000
        dc.l    $00000000,$00000000,$00000e00,$00700000
        dc.l    $00000000,$00000000,$00000700,$00e00000
        dc.l    $00000000,$00000000,$00000380,$01c00000
        dc.l    $00000000,$00000000,$000001c0,$03800000
```

```
        dc.l    $00000000,$00000000,$000000e0,$07000000
        dc.l    $00000000,$00000000,$00000070,$0e000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$0000001c,$38000000
        dc.l    $0000000f,$f0000000,$000f001f,$f800f000
        dc.l    $0000003f,$fc000000,$003f003f,$fc00fc00
        dc.l    $0000007f,$fe000000,$007c007f,$fe003e00
        dc.l    $000000ff,$ff000000,$00f800ff,$ff001f00
        dc.l    $000001ff,$ff800000,$01f001ff,$ff800f80
        dc.l    $000003ff,$ffc00000,$03e003ff,$ffc007c0
        dc.l    $000007ff,$ffe00000,$07c007ff,$ffe003e0
        dc.l    $00000fff,$fff00000,$0f800fff,$fff001f0
        dc.l    $00001fff,$fff80000,$1f001c3f,$fc3800f8
        dc.l    $00003fff,$fffc0000,$3e00381f,$f81c007c
        dc.l    $00003fff,$fffc0000,$7c00300f,$f00c003e
        dc.l    $00007fff,$fffe0000,$fc00700f,$f00e003f
        dc.l    $00007f8f,$f1fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff80f,$f01fffff
        dc.l    $00007f1f,$f8fe0000,$7ffffc1f,$f83ffffe
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00001fff,$fff80000,$00007fff,$fffe0000
        dc.l    $00001fff,$fff80000,$0000ffff,$ffff0000
        dc.l    $00000fff,$fff00000,$0001ff80,$01ff8000
        dc.l    $00000fff,$fff00000,$0003ffc0,$03ffc000
        dc.l    $000007ff,$ffe00000,$0007ffe0,$07ffe000
        dc.l    $000003ff,$ffc00000,$0007fff0,$0fffe000
        dc.l    $000001ff,$ff800000,$000ff1ff,$ff8ff000
        dc.l    $000001ff,$ff800000,$001fe1ff,$ff87f800
        dc.l    $000000ff,$ff000000,$003fc0ff,$ff03fc00
        dc.l    $0000007f,$fe000000,$003fc07f,$fe03fc00
        dc.l    $0000003f,$fc000000,$007f803f,$fc01fe00
        dc.l    $0000001f,$f8000000,$007f801f,$f801fe00
        dc.l    $0000000f,$f0000000,$00ff000f,$f000ff00
        dc.l    $00000003,$c0000000,$00ff0003,$c000ff00
        dc.l    $00000000,$00000000,$03fe0000,$00007fc0
        dc.l    $00000000,$00000000,$0ffe0000,$00007ff0
        dc.l    $00000000,$00000000,$3ffe0000,$00007ffc
        dc.l    $00000000,$00000000,$7fff0000,$0000fffe
        dc.l    $00000000,$00000000,$ffff0000,$0000ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff

        dc.l    0,0,0,0         ; Fine dello sprite (2 doppie longword).

        cnop    0,8

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end

Avete visto che bell'insettone largo 63 pixel? (ve lo sognate stanotte!)
La routine unimuovisprite e' stata modificata in modo molto semplice.
Infatti i 2 byte VSTOP e VHBITS si sono spostati una word + una long avanti.
```

```
Per cui e' bastato sostituire:

        2(a1) e 3(a1)

In:

        2+4+2(a1) e 3+4+2(a1)

Niente di piu' facile!
```

Lezione15f4

```
; Lezione15f4.s         Routine UniMuoviSprite fixata per fare lo scroll AGA
;                       orizzontale a step di 1/4 di pixel.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
        dbra    d1,NulLoop

;       Puntiamo lo sprite

        MOVE.L  #MIOSPRITE64,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
```

```
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

        move.b  $dff00a,mouse_y
        move.b  $dff00b,mouse_x

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        bsr.s   LeggiMouse      ; questa legge il mouse
        move.w  sprite_y(pc),d0 ; prepara i parametri per la routine
        move.w  sprite_x(pc),d1 ; universale
        lea     miosprite64,a1  ; indirizzo sprite
        moveq   #52,d2          ; altezza sprite
        bsr.w   UniMuoviSprite64F ; chiama la routine universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst.b  #2,$dff016      ; Tasto destro premuto?
        bne.s   NonScambiareRes ; Se no non cambiare risuluzione allo sprite

        bchg.b  #7,BplCon3      ; Se si, cambia da LowRes a Hires o viceversa.

NonScambiareRes:
        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b  $dff00a,d1      ; JOYODAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_y     ; modifica posizione sprite
no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
```

```
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_x     ; modifica pos. sprite
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

SPRITE_Y:       dc.w    0       ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.w    0       ; qui viene memorizzata la X dello sprite
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse


; Routine universale di posizionamento degli sprite larghi 64 pixel, con
; posizione X da 0 a 1280, che usa il nuovo scroll AGA ad 1/4 di pixel

;
;       Parametri in entrata di UniMuoviSprite64F:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-1280)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;       d2 = altezza dello sprite
;

UniMuoviSprite64F:
; posizionamento verticale
        ADD.W   #$2c,d0         ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b  d0,(a1)         ; copia il byte in VSTART
        btst.l  #8,d0
        beq.s   NonVSTARTSET
        bset.b  #2,3+4+2(a1)    ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,3+4+2(a1)    ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   D2,D0           ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,2+4+2(a1)    ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,3+4+2(a1)    ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,3+4+2(a1)    ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale - qua ci sono le modifiche!!!

        add.w   #128*4,D1               ; 128*4 - per centrare lo sprite.
        btst.l  #0,D1           ; bit basso 0 della coordinata X azzerato?
        beq.s   BitBassoZERO
        bset.b  #3,3+4+2(a1)    ; SH0 - Settiamo il bit piu' basso di HSTART
        bra.w   PlaceCoords1

BitBassoZERO:
        bclr.b  #3,3+4+2(a1)    ; Azzeriamo il bit basso di HSTART
PlaceCoords1:
        btst.l  #1,D1           ; bit basso 1 della coordinata X azzerato?
        beq.s   BitBassoZERO1
        bset.b  #4,3+4+2(a1)    ; SH1 - Settiamo il bit basso di HSTART
```

```
        bra.w   PlaceCoords2

BitBassoZERO1:
        bclr.b  #4,3+4+2(a1)    ; SH1 - Azzeriamo il bit basso di HSTART
PlaceCoords2:
        btst.l  #2,D1           ; bit basso 2 della coordinata X azzerato?
        beq.s   BitBassoZERO2
        bset.b  #0,3+4+2(a1)    ; SH2 - Settiamo il bit basso di HSTART
        bra.w   PlaceCoords3

BitBassoZERO2:
        bclr.b  #0,3+4+2(a1)    ; SH2 - Azzeriamo il bit basso di HSTART
PlaceCoords3:
        lsr.w   #3,D1           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; i 3 bit bassi.
        move.b  D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

;*****************************************************************************
;*                             COPPERLIST                                    *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,%1111      ; Burst mode a 64 bit, sprite larghi 64 pixel


BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.
```

```
        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w    $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w    $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO


        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$000       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$462       ; color17, nibble bassi
        dc.w    $1A4,$2e4       ; color18, nibble bassi
        dc.w    $1A6,$672       ; color19, nibble bassi

        dc.w    $106    ; BPLCON3
        dc.b    0
BplCon3:
              ; 76543210
        dc.b    %00000000       ; bit 7: sprites hires o lowres. Se si
                                ; settano sia il bit 7 che il bit 6 lo sprite
                                ; e' in superhires (1280x256), ma viene troppo
                                ; "stretto", credo sia inutile, basta hires!

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;***************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;***************************************************************************

        cnop    0,8

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
        dc.l    0,0,0,0         ; negli eventuali puntatori inutilizzati


        cnop    0,8

MIOSPRITE64:            ; lunghezza 13*4 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)
```

```
        dc.l    $00000000,$00000000,$00003000,$000c0000 ; Salvato con PicCon
        dc.l    $00000000,$00000000,$00003800,$001c0000
        dc.l    $00000000,$00000000,$00001c00,$00380000
        dc.l    $00000000,$00000000,$00000e00,$00700000
        dc.l    $00000000,$00000000,$00000700,$00e00000
        dc.l    $00000000,$00000000,$00000380,$01c00000
        dc.l    $00000000,$00000000,$000001c0,$03800000
        dc.l    $00000000,$00000000,$000000e0,$07000000
        dc.l    $00000000,$00000000,$00000070,$0e000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$0000001c,$38000000
        dc.l    $0000000f,$f0000000,$000f001f,$f800f000
        dc.l    $0000003f,$fc000000,$003f003f,$fc00fc00
        dc.l    $0000007f,$fe000000,$007c007f,$fe003e00
        dc.l    $000000ff,$ff000000,$00f800ff,$ff001f00
        dc.l    $000001ff,$ff800000,$01f001ff,$ff800f80
        dc.l    $000003ff,$ffc00000,$03e003ff,$ffc007c0
        dc.l    $000007ff,$ffe00000,$07c007ff,$ffe003e0
        dc.l    $00000fff,$fff00000,$0f800fff,$fff001f0
        dc.l    $00001fff,$fff80000,$1f001c3f,$fc3800f8
        dc.l    $00003fff,$fffc0000,$3e00381f,$f81c007c
        dc.l    $00003fff,$fffc0000,$7c00300f,$f00c003e
        dc.l    $00007fff,$fffe0000,$fc00700f,$f00e003f
        dc.l    $00007f8f,$f1fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff80f,$f01fffff
        dc.l    $00007f1f,$f8fe0000,$7ffffc1f,$f83ffffe
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00001fff,$fff80000,$00007fff,$fffe0000
        dc.l    $00001fff,$fff80000,$0000ffff,$ffff0000
        dc.l    $00000fff,$fff00000,$0001ff80,$01ff8000
        dc.l    $00000fff,$fff00000,$0003ffc0,$03ffc000
        dc.l    $000007ff,$ffe00000,$0007ffe0,$07ffe000
        dc.l    $000003ff,$ffc00000,$0007fff0,$0fffe000
        dc.l    $000001ff,$ff800000,$000ff1ff,$ff8ff000
        dc.l    $000001ff,$ff800000,$001fe1ff,$ff87f800
        dc.l    $000000ff,$ff000000,$003fc0ff,$ff03fc00
        dc.l    $0000007f,$fe000000,$003fc07f,$fe03fc00
        dc.l    $0000003f,$fc000000,$007f803f,$fc01fe00
        dc.l    $0000001f,$f8000000,$007f801f,$f801fe00
        dc.l    $0000000f,$f0000000,$00ff000f,$f000ff00
        dc.l    $00000003,$c0000000,$00ff0003,$c000ff00
        dc.l    $00000000,$00000000,$03fe0000,$00007fc0
        dc.l    $00000000,$00000000,$0ffe0000,$00007ff0
        dc.l    $00000000,$00000000,$3ffe0000,$00007ffc
        dc.l    $00000000,$00000000,$7fff0000,$0000fffe
        dc.l    $00000000,$00000000,$ffff0000,$0000ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff

        dc.l    0,0,0,0         ; Fine dello sprite (2 doppie longword).

        cnop    0,8

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
```

```
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

E' bastato aggiungere un paio di btst e il gioco e' fatto!

Lezione15f5

```
; Lezione15f5.s         Routine UniMuoviSprite fixata per fare lo scroll AGA
;                       orizzontale a step di 1/4 di pixel. Con una tabella
;                       rendiamo piu' evidente la maggiore fluidita'.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
        dbra    d1,NulLoop

;       Puntiamo lo sprite

        MOVE.L  #MIOSPRITE64,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
```

```
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        bsr.s   MuoviSpriteX    ; Legge dalla tabella e muove lo sprite

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst.b  #2,$dff016      ; Tasto destro premuto?
        bne.s   NonScambiareRes ; Se no non cambiare risuluzione allo sprite

        bchg.b  #7,BplCon3      ; Se si, cambia da LowRes a Hires o viceversa.

NonScambiareRes:
        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts

MuoviSpriteX:
        ADDQ.L  #2,TABXPOINT     ; Fai puntare alla word successiva
        MOVE.L  TABXPOINT(PC),A0 ; indirizzo contenuto in long TABXPOINT
                                 ; copiato in a0
        CMP.L   #FINETABX-2,A0  ; Siamo all'ultima word della TAB?
        BNE.S   NOBSTARTX       ; non ancora? allora continua
        MOVE.L  #TABX-2,TABXPOINT ; Riparti a puntare dalla prima word-2
NOBSTARTX:
        moveq   #0,d1           ; azzeriamo d0
        MOVE.w  (A0),d1         ; poniamo il valore della tabella in d0
;       add.w   #128*4,D1       ; 128*4 - per centrare lo sprite.
        moveq   #100,d0         ; Posizione Y
        moveq   #52,d2          ; altezza sprite
        lea     miosprite64,a1  ; indirizzo sprite
        bsr.w   UniMuoviSprite64F ; chiama la routine universale
        rts

TABXPOINT:
        dc.l    TABX-2          ; NOTA: i valori della tabella qua sono word,

; Ecco come ottenere la tabella:

;                                  ___1280
; DEST> tabx                      /   \ 640 (1280/2)
; BEG> 0                      \___/     0
; END> 360
; AMOUNT> 150*4
; AMPLITUDE> (1280-64*4)/2 ; ampiezza sia sopra zero che sotto zero, allora
                         ; bisogna che faccia meta' sopra zero e meta' sotto,
                         ; ossia dividiamo per 2 l'AMPIEZZA
```

```
; YOFFSET> (1280-64*4)/2 ; e spostiamo tutto sopra
; SIZE (B/W/L)> w
; MULTIPLIER> 1

TABX:
        DC.W    $0203,$0208,$020D,$0213,$0218,$021D,$0223,$0228,$022E,$0233
        DC.W    $0238,$023E,$0243,$0248,$024D,$0253,$0258,$025D,$0263,$0268
        DC.W    $026D,$0272,$0278,$027D,$0282,$0287,$028C,$0291,$0297,$029C
        DC.W    $02A1,$02A6,$02AB,$02B0,$02B5,$02BA,$02BF,$02C4,$02C9,$02CE
        DC.W    $02D3,$02D8,$02DC,$02E1,$02E6,$02EB,$02F0,$02F4,$02F9,$02FE
        DC.W    $0302,$0307,$030C,$0310,$0315,$0319,$031E,$0322,$0326,$032B
        DC.W    $032F,$0333,$0338,$033C,$0340,$0344,$0348,$034D,$0351,$0355
        DC.W    $0359,$035D,$0360,$0364,$0368,$036C,$0370,$0373,$0377,$037B
        DC.W    $037E,$0382,$0385,$0389,$038C,$0390,$0393,$0396,$0399,$039D
        DC.W    $03A0,$03A3,$03A6,$03A9,$03AC,$03AF,$03B2,$03B5,$03B7,$03BA
        DC.W    $03BD,$03BF,$03C2,$03C4,$03C7,$03C9,$03CC,$03CE,$03D0,$03D3
        DC.W    $03D5,$03D7,$03D9,$03DB,$03DD,$03DF,$03E1,$03E3,$03E4,$03E6
        DC.W    $03E8,$03E9,$03EB,$03EC,$03EE,$03EF,$03F1,$03F2,$03F3,$03F4
        DC.W    $03F5,$03F6,$03F7,$03F8,$03F9,$03FA,$03FB,$03FC,$03FC,$03FD
        DC.W    $03FD,$03FE,$03FE,$03FF,$03FF,$03FF,$0400,$0400,$0400,$0400
        DC.W    $0400,$0400,$0400,$0400,$03FF,$03FF,$03FF,$03FE,$03FE,$03FD
        DC.W    $03FD,$03FC,$03FC,$03FB,$03FA,$03F9,$03F8,$03F7,$03F6,$03F5
        DC.W    $03F4,$03F3,$03F2,$03F1,$03EF,$03EE,$03EC,$03EB,$03E9,$03E8
        DC.W    $03E6,$03E4,$03E3,$03E1,$03DF,$03DD,$03DB,$03D9,$03D7,$03D5
        DC.W    $03D3,$03D0,$03CE,$03CC,$03C9,$03C7,$03C4,$03C2,$03BF,$03BD
        DC.W    $03BA,$03B7,$03B5,$03B2,$03AF,$03AC,$03A9,$03A6,$03A3,$03A0
        DC.W    $039D,$0399,$0396,$0393,$0390,$038C,$0389,$0385,$0382,$037E
        DC.W    $037B,$0377,$0373,$0370,$036C,$0368,$0364,$0360,$035D,$0359
        DC.W    $0355,$0351,$034D,$0348,$0344,$0340,$033C,$0338,$0333,$032F
        DC.W    $032B,$0326,$0322,$031E,$0319,$0315,$0310,$030C,$0307,$0302
        DC.W    $02FE,$02F9,$02F4,$02F0,$02EB,$02E6,$02E1,$02DC,$02D8,$02D3
        DC.W    $02CE,$02C9,$02C4,$02BF,$02BA,$02B5,$02B0,$02AB,$02A6,$02A1
        DC.W    $029C,$0297,$0291,$028C,$0287,$0282,$027D,$0278,$0272,$026D
        DC.W    $0268,$0263,$025D,$0258,$0253,$024D,$0248,$0243,$023E,$0238
        DC.W    $0233,$022E,$0228,$0223,$021D,$0218,$0213,$020D,$0208,$0203
        DC.W    $01FD,$01F8,$01F3,$01ED,$01E8,$01E3,$01DD,$01D8,$01D2,$01CD
        DC.W    $01C8,$01C2,$01BD,$01B8,$01B3,$01AD,$01A8,$01A3,$019D,$0198
        DC.W    $0193,$018E,$0188,$0183,$017E,$0179,$0174,$016F,$0169,$0164
        DC.W    $015F,$015A,$0155,$0150,$014B,$0146,$0141,$013C,$0137,$0132
        DC.W    $012D,$0128,$0124,$011F,$011A,$0115,$0110,$010C,$0107,$0102
        DC.W    $00FE,$00F9,$00F4,$00F0,$00EB,$00E7,$00E2,$00DE,$00DA,$00D5
        DC.W    $00D1,$00CD,$00C8,$00C4,$00C0,$00BC,$00B8,$00B3,$00AF,$00AB
        DC.W    $00A7,$00A3,$00A0,$009C,$0098,$0094,$0090,$008D,$0089,$0085
        DC.W    $0082,$007E,$007B,$0077,$0074,$0070,$006D,$006A,$0067,$0063
        DC.W    $0060,$005D,$005A,$0057,$0054,$0051,$004E,$004B,$0049,$0046
        DC.W    $0043,$0041,$003E,$003C,$0039,$0037,$0034,$0032,$0030,$002D
        DC.W    $002B,$0029,$0027,$0025,$0023,$0021,$001F,$001D,$001C,$001A
        DC.W    $0018,$0017,$0015,$0014,$0012,$0011,$000F,$000E,$000D,$000C
        DC.W    $000B,$000A,$0009,$0008,$0007,$0006,$0005,$0004,$0004,$0003
        DC.W    $0003,$0002,$0002,$0001,$0001,$0001,$0000,$0000,$0000,$0000
        DC.W    $0000,$0000,$0000,$0000,$0001,$0001,$0001,$0002,$0002,$0003
        DC.W    $0003,$0004,$0004,$0005,$0006,$0007,$0008,$0009,$000A,$000B
        DC.W    $000C,$000D,$000E,$000F,$0011,$0012,$0014,$0015,$0017,$0018
        DC.W    $001A,$001C,$001D,$001F,$0021,$0023,$0025,$0027,$0029,$002B
        DC.W    $002D,$0030,$0032,$0034,$0037,$0039,$003C,$003E,$0041,$0043
        DC.W    $0046,$0049,$004B,$004E,$0051,$0054,$0057,$005A,$005D,$0060
        DC.W    $0063,$0067,$006A,$006D,$0070,$0074,$0077,$007B,$007E,$0082
        DC.W    $0085,$0089,$008D,$0090,$0094,$0098,$009C,$00A0,$00A3,$00A7
        DC.W    $00AB,$00AF,$00B3,$00B8,$00BC,$00C0,$00C4,$00C8,$00CD,$00D1
        DC.W    $00D5,$00DA,$00DE,$00E2,$00E7,$00EB,$00F0,$00F4,$00F9,$00FE
        DC.W    $0102,$0107,$010C,$0110,$0115,$011A,$011F,$0124,$0128,$012D
        DC.W    $0132,$0137,$013C,$0141,$0146,$014B,$0150,$0155,$015A,$015F
```

```
        DC.W    $0164,$0169,$016F,$0174,$0179,$017E,$0183,$0188,$018E,$0193
        DC.W    $0198,$019D,$01A3,$01A8,$01AD,$01B3,$01B8,$01BD,$01C2,$01C8
        DC.W    $01CD,$01D2,$01D8,$01DD,$01E3,$01E8,$01ED,$01F3,$01F8,$01FD
FINETABX:


; Routine universale di posizionamento degli sprite larghi 64 pixel, con
; posizione X da 0 a 1280, che usa il nuovo scroll AGA ad 1/4 di pixel

;
;       Parametri in entrata di UniMuoviSprite64F:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-1280)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;       d2 = altezza dello sprite
;

UniMuoviSprite64F:
; posizionamento verticale
        ADD.W   #$2c,d0         ; aggiungi l'offset dell'inizio dello schermo

; a1 contiene l'indirizzo dello sprite
        MOVE.b  d0,(a1)         ; copia il byte in VSTART
        btst.l  #8,d0
        beq.s   NonVSTARTSET
        bset.b  #2,3+4+2(a1)    ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,3+4+2(a1)    ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   D2,D0           ; Aggiungi l'altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,2+4+2(a1)    ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,3+4+2(a1)    ; Setta il bit 8 di VSTOP (numero > $FF)
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,3+4+2(a1)    ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale - qua ci sono le modifiche!!!

        add.w   #128*4,D1               ; 128*4 - per centrare lo sprite.
        btst.l  #0,D1           ; bit basso 0 della coordinata X azzerato?
        beq.s   BitBassoZERO
        bset.b  #3,3+4+2(a1)    ; SH0 - Settiamo il bit piu' basso di HSTART
        bra.w   PlaceCoords1

BitBassoZERO:
        bclr.b  #3,3+4+2(a1)    ; Azzeriamo il bit basso di HSTART
PlaceCoords1:
        btst.l  #1,D1           ; bit basso 1 della coordinata X azzerato?
        beq.s   BitBassoZERO1
        bset.b  #4,3+4+2(a1)    ; SH1 - Settiamo il bit basso di HSTART
        bra.w   PlaceCoords2

BitBassoZERO1:
        bclr.b  #4,3+4+2(a1)    ; SH1 - Azzeriamo il bit basso di HSTART
PlaceCoords2:
        btst.l  #2,D1           ; bit basso 2 della coordinata X azzerato?
```

```
        beq.s   BitBassoZERO2
        bset.b  #0,3+4+2(a1)    ; SH2 - Settiamo il bit basso di HSTART
        bra.w   PlaceCoords3

BitBassoZERO2:
        bclr.b  #0,3+4+2(a1)    ; SH2 - Azzeriamo il bit basso di HSTART
PlaceCoords3:
        lsr.w   #3,D1           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; i 3 bit bassi.
        move.b  D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,%1111      ; Burst mode a 64 bit, sprite larghi 64 pixel


BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
```

```
                                        ; per cui non compare.

        dc.w    $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w    $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w    $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO


        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$000       ; color1        ; colore 1 del bitplane, che
                                        ; in questo caso e' vuoto,
                                        ; per cui non compare.

        dc.w    $1A2,$462       ; color17, nibble bassi
        dc.w    $1A4,$2e4       ; color18, nibble bassi
        dc.w    $1A6,$672       ; color19, nibble bassi

        dc.w    $106    ; BPLCON3
        dc.b    0
BplCon3:
               ; 76543210
        dc.b    %00000000       ; bit 7: sprites hires o lowres. Se si
                                ; settano sia il bit 7 che il bit 6 lo sprite
                                ; e' in superhires (1280x256), ma viene troppo
                                ; "stretto", credo sia inutile, basta hires!

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;*****************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;*****************************************************************************

        cnop    0,8

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
        dc.l    0,0,0,0         ; negli eventuali puntatori inutilizzati


        cnop    0,8

MIOSPRITE64:            ; lunghezza 13*4 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)

        dc.l    $00000000,$00000000,$00003000,$000c0000 ; Salvato con PicCon
        dc.l    $00000000,$00000000,$00003800,$001c0000
        dc.l    $00000000,$00000000,$00001c00,$00380000
        dc.l    $00000000,$00000000,$00000e00,$00700000
        dc.l    $00000000,$00000000,$00000700,$00e00000
        dc.l    $00000000,$00000000,$00000380,$01c00000
```

```
        dc.l    $00000000,$00000000,$000001c0,$03800000
        dc.l    $00000000,$00000000,$000000e0,$07000000
        dc.l    $00000000,$00000000,$00000070,$0e000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$0000001c,$38000000
        dc.l    $0000000f,$f0000000,$000f001f,$f800f000
        dc.l    $0000003f,$fc000000,$003f003f,$fc00fc00
        dc.l    $0000007f,$fe000000,$007c007f,$fe003e00
        dc.l    $000000ff,$ff000000,$00f800ff,$ff001f00
        dc.l    $000001ff,$ff800000,$01f001ff,$ff800f80
        dc.l    $000003ff,$ffc00000,$03e003ff,$ffc007c0
        dc.l    $000007ff,$ffe00000,$07c007ff,$ffe003e0
        dc.l    $00000fff,$fff00000,$0f800fff,$fff001f0
        dc.l    $00001fff,$fff80000,$1f001c3f,$fc3800f8
        dc.l    $00003fff,$fffc0000,$3e00381f,$f81c007c
        dc.l    $00003fff,$fffc0000,$7c00300f,$f00c003e
        dc.l    $00007fff,$fffe0000,$fc00700f,$f00e003f
        dc.l    $00007f8f,$f1fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff80f,$f01fffff
        dc.l    $00007f1f,$f8fe0000,$7ffffc1f,$f83ffffe
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00001fff,$fff80000,$00007fff,$fffe0000
        dc.l    $00001fff,$fff80000,$0000ffff,$ffff0000
        dc.l    $00000fff,$fff00000,$0001ff80,$01ff8000
        dc.l    $00000fff,$fff00000,$0003ffc0,$03ffc000
        dc.l    $000007ff,$ffe00000,$0007ffe0,$07ffe000
        dc.l    $000003ff,$ffc00000,$0007fff0,$0fffe000
        dc.l    $000001ff,$ff800000,$000ff1ff,$ff8ff000
        dc.l    $000001ff,$ff800000,$001fe1ff,$ff87f800
        dc.l    $000000ff,$ff000000,$003fc0ff,$ff03fc00
        dc.l    $0000007f,$fe000000,$003fc07f,$fe03fc00
        dc.l    $0000003f,$fc000000,$007f803f,$fc01fe00
        dc.l    $0000001f,$f8000000,$007f801f,$f801fe00
        dc.l    $0000000f,$f0000000,$00ff000f,$f000ff00
        dc.l    $00000003,$c0000000,$00ff0003,$c000ff00
        dc.l    $00000000,$00000000,$03fe0000,$00007fc0
        dc.l    $00000000,$00000000,$0ffe0000,$00007ff0
        dc.l    $00000000,$00000000,$3ffe0000,$00007ffc
        dc.l    $00000000,$00000000,$7fff0000,$0000fffe
        dc.l    $00000000,$00000000,$ffff0000,$0000ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff

        dc.l    0,0,0,0         ; Fine dello sprite (2 doppie longword).

        cnop    0,8

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

Da notare che uno scroll del genere non e' possibile su PC MSDOS, e neanche
sul super nintendo, per non parlare del sega megadrive...

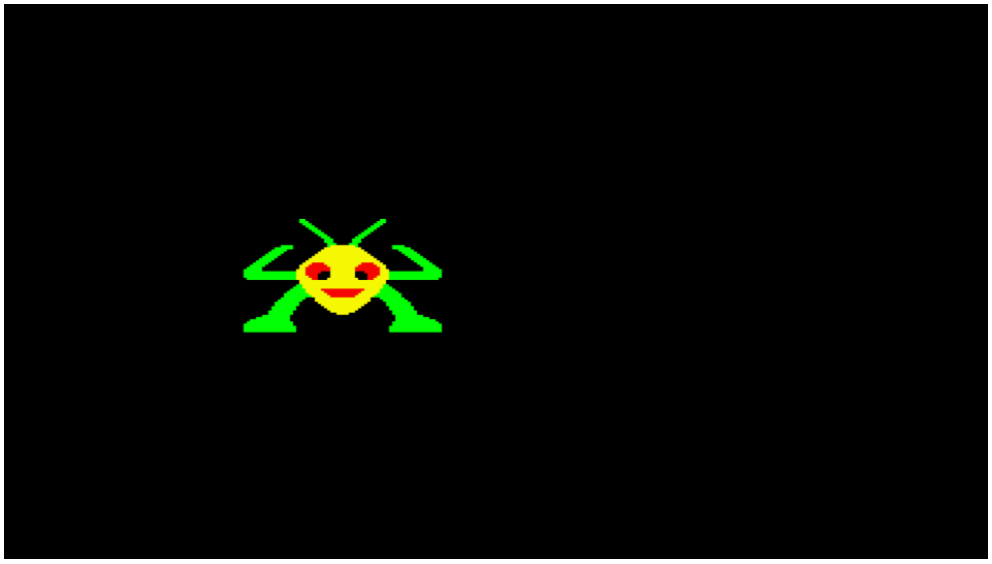


Figura 29.5: Lezione 15f5

Lezione15f6

```
; Lezione15f6.s         Premere il tasto destro per attivare BRDRSPRT, che
;                       permette la visualizzazione dello sprite fuori dalla
;                       finestra video, definita di proposito piu' stretta
;                       del normale.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo tutti gli sprite allo sprite nullo

        MOVE.L  #SpriteNullo,d0         ; indirizzo dello sprite in d0
```

```
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        MOVEQ   #8-1,d1                 ; tutti gli 8 sprite
NulLoop:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addq.w  #8,a1
        dbra    d1,NulLoop

;       Puntiamo lo sprite

        MOVE.L  #MIOSPRITE64,d0         ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

        move.b  $dff00a,mouse_y
        move.b  $dff00b,mouse_x

mouse:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BNE.S   Waity1

        bsr.s   LeggiMouse      ; questa legge il mouse
        move.w  sprite_y(pc),d0 ; prepara i parametri per la routine
        move.w  sprite_x(pc),d1 ; universale
        lea     miosprite64,a1  ; indirizzo sprite
        moveq   #52,d2          ; altezza sprite
        bsr.w   UniMuoviSprite64F ; chiama la routine universale

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$12000,d2      ; linea da aspettare = $120
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $120
        BEQ.S   Aspetta

        btst.b  #2,$dff016      ; Tasto destro premuto?
        bne.s   NonScambiareRes ; Se no non cambiare

        bchg.b  #1,BplCon3      ; Se si, cambia il BRDRSPRT.

NonScambiareRes:
        btst.b  #6,$bfe001      ; mouse premuto?
        bne.s   mouse
        rts
```

```
; Questa routine legge il mouse e aggiorna i valori contenuti nelle
; variabili sprite_x e sprite_y

LeggiMouse:
        move.b  $dff00a,d1      ; JOYODAT posizione verticale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_y(PC),d0  ; sottrai vecchia posizione mouse
        beq.s   no_vert         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_y     ; modifica posizione sprite
no_vert:
        move.b  d1,mouse_y      ; salva posizione mouse per la prossima volta

        move.b  $dff00b,d1      ; posizione orizzontale mouse
        move.b  d1,d0           ; copia in d0
        sub.b   mouse_x(PC),d0  ; sottrai vecchia posizione
        beq.s   no_oriz         ; se la differenza = 0, il mouse e‘ fermo
        ext.w   d0              ; trasforma il byte in word
                                ; (vedi alla fine del listato)
        add.w   d0,sprite_x     ; modifica pos. sprite
no_oriz
        move.b  d1,mouse_x      ; salva posizione mouse per la prossima volta
        RTS

SPRITE_Y:       dc.w    0       ; qui viene memorizzata la Y dello sprite
SPRITE_X:       dc.w    0       ; qui viene memorizzata la X dello sprite
MOUSE_Y:        dc.b    0       ; qui viene memorizzata la Y del mouse
MOUSE_X:        dc.b    0       ; qui viene memorizzata la X del mouse


; Routine universale di posizionamento degli sprite larghi 64 pixel, con
; posizione X da 0 a 1280, che usa il nuovo scroll AGA ad 1/4 di pixel

;
;       Parametri in entrata di UniMuoviSprite64F:
;
;       a1 = Indirizzo dello sprite
;       d0 = posizione verticale Y dello sprite sullo schermo (0-1280)
;       d1 = posizione orizzontale X dello sprite sullo schermo (0-320)
;       d2 = altezza dello sprite
;

UniMuoviSprite64F:
; posizionamento verticale
        ADD.W   #$2c,d0         ; aggiungi l’offset dell’inizio dello schermo

; a1 contiene l’indirizzo dello sprite
        MOVE.b  d0,(a1)         ; copia il byte in VSTART
        btst.l  #8,d0
        beq.s   NonVSTARTSET
        bset.b  #2,3+4+2(a1)    ; Setta il bit 8 di VSTART (numero > $FF)
        bra.s   ToVSTOP
NonVSTARTSET:
        bclr.b  #2,3+4+2(a1)    ; Azzera il bit 8 di VSTART (numero < $FF)
ToVSTOP:
        ADD.w   D2,D0           ; Aggiungi l’altezza dello sprite per
                                ; determinare la posizione finale (VSTOP)
        move.b  d0,2+4+2(a1)    ; Muovi il valore giusto in VSTOP
        btst.l  #8,d0
        beq.s   NonVSTOPSET
        bset.b  #1,3+4+2(a1)    ; Setta il bit 8 di VSTOP (numero > $FF)
```

```
        bra.w   VstopFIN
NonVSTOPSET:
        bclr.b  #1,3+4+2(a1)    ; Azzera il bit 8 di VSTOP (numero < $FF)
VstopFIN:

; posizionamento orizzontale - qua ci sono le modifiche!!!

        add.w   #128*4,D1               ; 128*4 - per centrare lo sprite.
        btst.l  #0,D1           ; bit basso 0 della coordinata X azzerato?
        beq.s   BitBassoZERO
        bset.b  #3,3+4+2(a1)    ; SH0 - Settiamo il bit piu' basso di HSTART
        bra.w   PlaceCoords1

BitBassoZERO:
        bclr.b  #3,3+4+2(a1)    ; Azzeriamo il bit basso di HSTART
PlaceCoords1:
        btst.l  #1,D1           ; bit basso 1 della coordinata X azzerato?
        beq.s   BitBassoZERO1
        bset.b  #4,3+4+2(a1)    ; SH1 - Settiamo il bit basso di HSTART
        bra.w   PlaceCoords2

BitBassoZERO1:
        bclr.b  #4,3+4+2(a1)    ; SH1 - Azzeriamo il bit basso di HSTART
PlaceCoords2:
        btst.l  #2,D1           ; bit basso 2 della coordinata X azzerato?
        beq.s   BitBassoZERO2
        bset.b  #0,3+4+2(a1)    ; SH2 - Settiamo il bit basso di HSTART
        bra.w   PlaceCoords3

BitBassoZERO2:
        bclr.b  #0,3+4+2(a1)    ; SH2 - Azzeriamo il bit basso di HSTART
PlaceCoords3:
        lsr.w   #3,D1           ; SHIFTIAMO, ossia spostiamo di 1 bit a destra
                                ; il valore di HSTART, per "trasformarlo" nel
                                ; valore fa porre nel byte HSTART, senza cioe'
                                ; i 3 bit bassi.
        move.b  D1,1(a1)        ; Poniamo il valore XX nel byte HSTART
        rts

;*****************************************************************************
;*                             COPPERLIST                                    *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$5c81       ; DiwStrt
        dc.w    $90,$fcc1       ; DiwStop

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
```

```
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,%1111      ; Burst mode a 64 bit, sprite larghi 64 pixel


BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$F00       ; color17, ossia COLOR1 dello sprite0 - ROSSO
        dc.w    $1A4,$0F0       ; color18, ossia COLOR2 dello sprite0 - VERDE
        dc.w    $1A6,$FF0       ; color19, ossia COLOR3 dello sprite0 - GIALLO


        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$000       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        dc.w    $1A2,$462       ; color17, nibble bassi
        dc.w    $1A4,$2e4       ; color18, nibble bassi
        dc.w    $1A6,$672       ; color19, nibble bassi

        dc.w    $106    ; BPLCON3
        dc.b    0
BplCon3:
               ; 76543210
        dc.b    %00000000       ; bit 7: sprites hires o lowres. Se si
                                ; settano sia il bit 7 che il bit 6 lo sprite
                                ; e' in superhires (1280x256), ma viene troppo
                                ; "stretto", credo sia inutile, basta hires!

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;***************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;***************************************************************************

        cnop    0,8

SpriteNullo:                    ; Sprite nullo da puntare in copperlist
```

```
        dc.l    0,0,0,0         ; negli eventuali puntatori inutilizzati


        cnop    0,8

MIOSPRITE64:            ; lunghezza 13*4 linee
VSTART:
        dc.b $50        ; Posizione verticale di inizio sprite (da $2c a $f2)
HSTART:
        dc.b $90        ; Posizione orizzontale di inizio sprite (da $40 a $d8)
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)
VSTOP:
        dc.b $5d        ; $50+13=$5d    ; posizione verticale di fine sprite
VHBITS:
        dc.b $00        ; bit
        dc.w 0          ; Word + longword aggiunte per raggiungere la doppia
        dc.l 0          ; longword nello sprite largo 64 pixel (2 long!)

        dc.l    $00000000,$00000000,$00003000,$000c0000 ; Salvato con PicCon
        dc.l    $00000000,$00000000,$00003800,$001c0000
        dc.l    $00000000,$00000000,$00001c00,$00380000
        dc.l    $00000000,$00000000,$00000e00,$00700000
        dc.l    $00000000,$00000000,$00000700,$00e00000
        dc.l    $00000000,$00000000,$00000380,$01c00000
        dc.l    $00000000,$00000000,$000001c0,$03800000
        dc.l    $00000000,$00000000,$000000e0,$07000000
        dc.l    $00000000,$00000000,$00000070,$0e000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$00000038,$1c000000
        dc.l    $00000000,$00000000,$0000001c,$38000000
        dc.l    $0000000f,$f0000000,$000f001f,$f800f000
        dc.l    $0000003f,$fc000000,$003f003f,$fc00fc00
        dc.l    $0000007f,$fe000000,$007c007f,$fe003e00
        dc.l    $000000ff,$ff000000,$00f800ff,$ff001f00
        dc.l    $000001ff,$ff800000,$01f001ff,$ff800f80
        dc.l    $000003ff,$ffc00000,$03e003ff,$ffc007c0
        dc.l    $000007ff,$ffe00000,$07c007ff,$ffe003e0
        dc.l    $00000fff,$fff00000,$0f800fff,$fff001f0
        dc.l    $00001fff,$fff80000,$1f001c3f,$fc3800f8
        dc.l    $00003fff,$fffc0000,$3e00381f,$f81c007c
        dc.l    $00003fff,$fffc0000,$7c00300f,$f00c003e
        dc.l    $00007fff,$fffe0000,$fc00700f,$f00e003f
        dc.l    $00007f8f,$f1fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff00f,$f00fffff
        dc.l    $00007f0f,$f0fe0000,$fffff80f,$f01fffff
        dc.l    $00007f1f,$f8fe0000,$7ffffc1f,$f83ffffe
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00003fff,$fffc0000,$00003fff,$fffc0000
        dc.l    $00001fff,$fff80000,$00007fff,$fffe0000
        dc.l    $00001fff,$fff80000,$0000ffff,$ffff0000
        dc.l    $00000fff,$fff00000,$0001ff80,$01ff8000
        dc.l    $00000fff,$fff00000,$0003ffc0,$03ffc000
        dc.l    $000007ff,$ffe00000,$0007ffe0,$07ffe000
        dc.l    $000003ff,$ffc00000,$0007fff0,$0fffe000
        dc.l    $000001ff,$ff800000,$000ff1ff,$ff8ff000
        dc.l    $000001ff,$ff800000,$001fe1ff,$ff87f800
        dc.l    $000000ff,$ff000000,$003fc0ff,$ff03fc00
        dc.l    $0000007f,$fe000000,$003fc07f,$fe03fc00
        dc.l    $0000003f,$fc000000,$007f803f,$fc01fe00
        dc.l    $0000001f,$f8000000,$007f801f,$f801fe00
        dc.l    $0000000f,$f0000000,$00ff000f,$f000ff00
```

```
        dc.l    $00000003,$c0000000,$00ff0003,$c000ff00
        dc.l    $00000000,$00000000,$03fe0000,$00007fc0
        dc.l    $00000000,$00000000,$0ffe0000,$00007ff0
        dc.l    $00000000,$00000000,$3ffe0000,$00007ffc
        dc.l    $00000000,$00000000,$7fff0000,$0000fffe
        dc.l    $00000000,$00000000,$ffff0000,$0000ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff
        dc.l    $00000000,$00000000,$ffff8000,$0001ffff

        dc.l    0,0,0,0         ; Fine dello sprite (2 doppie longword).

        cnop    0,8

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

Basta settare il bit magico, e gli sprite si possono muovere veramente in modo indipendente dallo schermo, dato che non vengono "tagliati" dalla fine della finestra video, e possono essere visualizzata anche se i bitplanes sono disabilitati! Una liberazione degli sprites dalla schiavitu' del bplcon0 e dei diwstart/diwstop!

Lezione15f7

```
; Lezione15f7.s         Palette degli sprite AGA spostata con $dff10c.
;                       Tasto sinistro per assegnare 2 palette diverse
;                       agli sprite pari e dispari, destro per uscire.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110100000       ; copper, bitplane, sprite DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la PIC "vuota"

        MOVE.L  #BITPLANE,d0    ; dove puntare
        LEA     BPLPOINTERS,A1  ; puntatori COP
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo gli sprite
```

```
        MOVE.L  #MIOSPRITE0,d0          ; indirizzo dello sprite in d0
        LEA     SpritePointers,a1       ; Puntatori in copperlist
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE1,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE2,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE3,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE4,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE5,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE6,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        MOVE.L  #MIOSPRITE7,d0          ; indirizzo dello sprite in d0
        addq.w  #8,a1                   ; prossimi SPRITEPOINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

        move.w  #%11101110,bplcon4+2    ; stessa palette per sprite pari
                                        ; e dispari

mouseS:
        btst.b  #6,$bfe001      ; mouse sin. premuto?
        bne.s   mouseS

        move.w  #%11101111,bplcon4+2    ; palette diversa per sprite pari
                                        ; e dispari

mouseD:
        btst.b  #2,$dff016      ; mouse dest. premuto?
        bne.s   mouseD
```

```
        rts

;******************************************************************************
;*                              COPPERLIST                                    *
;******************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
SpritePointers:
        dc.w    $120,0,$122,0,$124,0,$126,0,$128,0 ; SPRITE
        dc.w    $12a,0,$12c,0,$12e,0,$130,0,$132,0
        dc.w    $134,0,$136,0,$138,0,$13a,0,$13c,0
        dc.w    $13e,0

        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LORES 320x256.

        dc.w    $1fc,%0011      ; Burst mode a 64 bit, sprite larghi 16 pixel


BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; Da notare che i colori dello sprite sono a 24 bit, anche se sono solo 3.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        dc.w    $180,$000       ; color0        ; sfondo nero
        dc.w    $182,$123       ; color1        ; colore 1 del bitplane, che
                                                ; in questo caso e' vuoto,
                                                ; per cui non compare.

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

bplcon4:
        dc.w    $10c,%11101111  ; BPLCON4 - Palette sprite pari = 224-240
                                ;           Palette sprite dispari = 240-256

; Ora il banco della palette sprite pari

        dc.w    $182,$F00       ; color225, - COLOR1 dello sprite0 -ROSSO
        dc.w    $184,$0F0       ; color226, - COLOR2 dello sprite0 -VERDE
```

```
        dc.w    $186,$FF0       ; color227, - COLOR3 dello sprite0 -GIALLO

        dc.w    $18A,$FFF       ; color229, - COLOR1 dello sprite2 -BIANCO
        dc.w    $18C,$0BD       ; color230, - COLOR2 dello sprite2 -ACQUA
        dc.w    $18E,$D50       ; color231, - COLOR3 dello sprite2 -ARANCIO

        dc.w    $192,$00F       ; color233, - COLOR1 dello sprite4 -BLU
        dc.w    $194,$F0F       ; color234, - COLOR2 dello sprite4 -VIOLA
        dc.w    $196,$BBB       ; color235, - COLOR3 dello sprite4 -GRIGIO

        dc.w    $19A,$8E0       ; color237, - COLOR1 dello sprite6 -VERDE CH.
        dc.w    $19C,$a70       ; color238, - COLOR2 dello sprite6 -MARRONE
        dc.w    $19E,$d00       ; color239, - COLOR3 dello sprite6 -ROSSO SC.

; Ora il banco della palette sprite dispari

        dc.w    $1A2,$555       ; color241, - COLOR1 dello sprite1 -grigio
        dc.w    $1A4,$aa0       ; color242, - COLOR2 dello sprite1 -giallo
        dc.w    $1A6,$0af       ; color243, - COLOR3 dello sprite1 -azzurro

        dc.w    $1AA,$a0a       ; color245, - COLOR1 dello sprite3 -...
        dc.w    $1AC,$3fa       ; color246, - COLOR2 dello sprite3 -...
        dc.w    $1AE,$faf       ; color247, - COLOR3 dello sprite3 -...

        dc.w    $1B2,$254       ; color249, - COLOR1 dello sprite5 -...
        dc.w    $1B4,$5a3       ; color250, - COLOR2 dello sprite5 -...
        dc.w    $1B6,$4ee       ; color251, - COLOR3 dello sprite5 -...

        dc.w    $1BA,$22c       ; color253, - COLOR1 dello sprite7 -...
        dc.w    $1BC,$381       ; color354, - COLOR2 dello sprite7 -...
        dc.w    $1BE,$fe9       ; color255, - COLOR3 dello sprite7 -...

        dc.w    $FFFF,$FFFE     ; Fine della copperlist


;***************************************************************************
;************ Ecco gli sprite: OVVIAMENTE devono essere in CHIP RAM! *********
;***************************************************************************

MIOSPRITE0:             ; lunghezza 13 linee
VSTART0:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART0:
        dc.b $60        ; Pos. orizzontale (da $40 a $d8)
VSTOP0:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111001110001111
 dc.w   %0011111111111100,%1100010001000011
 dc.w   %0111111111111110,%1000010001000001
 dc.w   %0111111111111110,%1000010001000001
 dc.w   %0011111111111100,%1100010001000011
 dc.w   %0000111111110000,%1111001110001111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite


MIOSPRITE1:             ; lunghezza 13 linee
VSTART1:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART1:
```

```
        dc.b $60+14     ; Pos. orizzontale (da $40 a $d8)
VSTOP1:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111000010001111
 dc.w   %0011111111111100,%1100000110000011
 dc.w   %0111111111111110,%1000000010000001
 dc.w   %0111111111111110,%1000000010000001
 dc.w   %0011111111111100,%1100000010000011
 dc.w   %0000111111110000,%1111000111001111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

 ; per gli sprite 2 e 3
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BIANCO)
 ;BINARIO 01=COLORE 2 (ACQUA)
 ;BINARIO 11=COLORE 3 (ARANCIO)

MIOSPRITE2:             ; lunghezza 13 linee
VSTART2:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART2:
        dc.b $60+(14*2) ; Pos. orizzontale (da $40 a $d8)
VSTOP2:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111000111001111
 dc.w   %0011111111111100,%1100001000100011
 dc.w   %0111111111111110,%1000000000100001
 dc.w   %0111111111111110,%1000000111000001
 dc.w   %0011111111111100,%1100001000000011
 dc.w   %0000111111110000,%1111001111101111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

MIOSPRITE3:             ; lunghezza 13 linee
VSTART3:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART3:
        dc.b $60+(14*3) ; Pos. orizzontale (da $40 a $d8)
VSTOP3:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111001111101111
 dc.w   %0011111111111100,%1100000000100011
 dc.w   %0111111111111110,%1000000111100001
 dc.w   %0111111111111110,%1000000000100001
 dc.w   %0011111111111100,%1100000000100011
 dc.w   %0000111111110000,%1111001111101111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

 ; per gli sprite 4 e 5
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (BLU)
 ;BINARIO 01=COLORE 2 (VIOLA)
 ;BINARIO 11=COLORE 3 (GRIGIO)
```

```
MIOSPRITE4:             ; lunghezza 13 linee
VSTART4:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART4:
        dc.b $60+(14*4) ; Pos. orizzontale (da $40 a $d8)
VSTOP4:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111001001001111
 dc.w   %0011111111111100,%1100001001000011
 dc.w   %0111111111111110,%1000001111000001
 dc.w   %0111111111111110,%1000000001000001
 dc.w   %0011111111111100,%1100000001000011
 dc.w   %0000111111110000,%1111000001001111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

MIOSPRITE5:             ; lunghezza 13 linee
VSTART5:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART5:
        dc.b $60+(14*5) ; Pos. orizzontale (da $40 a $d8)
VSTOP5:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111001111001111
 dc.w   %0011111111111100,%1100001000000011
 dc.w   %0111111111111110,%1000001111000001
 dc.w   %0111111111111110,%1000000001000001
 dc.w   %0011111111111100,%1100000001000011
 dc.w   %0000111111110000,%1111001111001111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

 ; per gli sprite 6 e 7
 ;BINARIO 00=COLORE 0 (TRASPARENTE)
 ;BINARIO 10=COLORE 1 (VERDE CHIARO)
 ;BINARIO 01=COLORE 2 (MARRONE)
 ;BINARIO 11=COLORE 3 (ROSSO SCURO)

MIOSPRITE6:             ; lunghezza 13 linee
VSTART6:
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART6:
        dc.b $60+(14*6) ; Pos. orizzontale (da $40 a $d8)
VSTOP6:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111001111001111
 dc.w   %0011111111111100,%1100001000000011
 dc.w   %0111111111111110,%1000001111000001
 dc.w   %0111111111111110,%1000001001000001
 dc.w   %0011111111111100,%1100001001000011
 dc.w   %0000111111110000,%1111001111001111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

MIOSPRITE7:             ; lunghezza 13 linee
VSTART7:
```

```
        dc.b $60        ; Pos. verticale (da $2c a $f2)
HSTART7:
        dc.b $60+(14*7) ; Pos. orizzontale (da $40 a $d8)
VSTOP7:
        dc.b $68        ; $60+13=$6d    ; fine verticale.
        dc.b $00
 dc.w   %0000001111000000,%0111110000111110
 dc.w   %0000111111110000,%1111001111001111
 dc.w   %0011111111111100,%1100000001000011
 dc.w   %0111111111111110,%1000000001000001
 dc.w   %0111111111111110,%1000000001000001
 dc.w   %0011111111111100,%1100000001000011
 dc.w   %0000111111110000,%1111000001001111
 dc.w   %0000001111000000,%0111110000111110
 dc.w   0,0     ; fine sprite

        SECTION PLANEVUOTO,BSS_C        ; Il bitplane azzerato che usiamo,
                                        ; perche' per vedere gli sprite
                                        ; e' necessario che ci siano bitplanes
                                        ; abilitati
BITPLANE:
        ds.b    40*256          ; bitplane azzerato lowres

        end
```

Tutto il listato e' basato su questi due registri in copperlist:

```
        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        dc.w    $10c,%11101111  ; BPLCON4 - Palette sprite pari = 224-240
                                ;           Palette sprite dispari = 240-256
```

Poi vengono settati i colori dal 225 a 255 (solo i nibble alti, non avevo voglia di mettere anche quelli bassi!).

"Spostare" la palette degli sprite in fondo alla palette puo' essere utile nel caso si visualizzino figure fino a 128 colori, per cui la palette dei nostri sprite e' totalmente indipendente. Nel caso la figura sia a 256 colori, si puo' optare per un qualsiasi banco di 16 colori usabili anche per gli sprite.

29.7 Lezione15g

```
; Lezione15g1.s         - Scroll AGA dei bitplanes a scatti di 1/4 di pixel,
;                         per un messimo di 64 pixel.

; NOTA: I 2 bit alti dello scroll, che permettono "scatti" di 16 o di 32 pixel,
; per un massimo di 64 pixel di scroll, funzionano solo se il burst mode e'
; a 64 pixel (settando i 2 bit bassi di FMODE, ossia $dff1fc).

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA
```

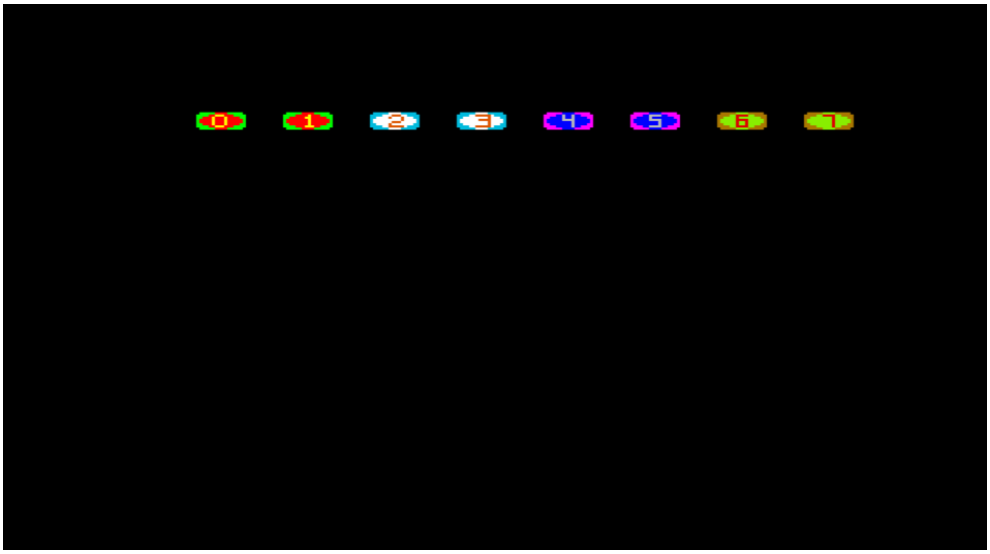


Figura 29.6: Lezione 15f7

```
WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PIC1,d0
        LEA     EVENBPLPT,A1            ; BPL POINTERS
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

        bsr.w   MAKEMOVTAB      ; Questa semplice routine fa una tabella
                                ; con valori da 0 a 255, poi di nuovo a 0

        bsr.w   FINESCROLLC     ; Questa routine "converte" i valori decimali
                                ; in valori di scroll per il BPLCON1 AGA

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #AgaCopList,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1
```

```
        BSR.w   WABBLE

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

*****************************************************************************
; Questa routine fa una tabella con valori "decimali" 0-255-0, in una tabella
*****************************************************************************

NUMVAL = 255    ; Il nuovo bplcon1 puo' andare da 0 a 255, sfruttiamolo!

makemovtab:
        LEA     MOVTAB(PC),A0   ; Tab valori playfield 1
        MOVEQ   #0,D0           ; MINIMO PLAYFIELD 1 : 0
        MOVE.L  #NUMVAL,D1      ; MASSIMO : numval
AMTLOOP:
        MOVE.W  D0,(A0)+        ; Val playfield 1
        ADDQ.L  #1,D0           ; aggiungi 1 al val. playfield 1
        CMP.L   D1,D0           ; pf1=numval? (allora pf2=zero)
        BNE.S   AMTLOOP         ; se non ancora, continua a loopare.
AMTLOOP2:
        MOVE.W  D0,(A0)+        ; Val Pf1 - (Da numval a 0)
        SUBQ.L  #1,D0           ; Subba val. Pf1
        BNE.S   AMTLOOP2        ; d0=zero? (flag Z) - se non ancora loopa!
        RTS

MOVTAB:
        DCB.W   NUMVAL*2,0      ; *2 perche' sono words
MOVTABEND:


*****************************************************************************
; Routine che converte da numeri "decimali" a valori per il bplcon1 AGA.
; In pratica scompone il numero a 8 bit posizionando i sui bit secondo lo
; schema del bplcon1 aga:
;
;       15      64 PIXEL SCROLL PF2 (AGA)
;       14      64 PIXEL SCROLL PF2 (AGA)
;       13      FINE SCROLL PF2 (AGA SCROLL 35ns 1/4 of pixel)
;       12      FINE SCROLL PF2
;       11      64 PIXEL SCROLL PF1 (AGA)
;       10      64 PIXEL SCROLL PF1 (AGA)
;       09      FINE SCROLL PF1 (AGA SCROLL 35ns 1/4 of pixel)
;       08      FINE SCROLL PF1
;       07      PF2H3
;       06      PF2H2
;       05      PF2H1
;       04      PF2H0
;       03      PF1H3
;       02      PF1H2
;       01      PF1H1
;       00      PF1H0
```

```
*****************************************************************************

FINESCROLLC:
        LEA     MOVTAB(PC),A0           ; Tab valori playfield 2
        LEA     CON1VALUES(PC),A1       ; Tab destinazione per $DFF002
        LEA     MOVTABEND(PC),a2        ; Fine della tabella
CONVLOOP:
        MOVEQ   #0,D1
        MOVE.W  (A0)+,D1        ; VALORE "DECIMALE" PF1 IN D1
        MOVE.W  D1,D2           ; COPIA VAL. 1 IN D2
        MOVE.W  d1,d4           ; COPIA VAL. 1 IN D4
;
        AND.W   #%11,D1         ; Selez. bits 0-1 (SCROLL 1/4 e 1/2 pixel)
        LSL.W   #8,D1           ; Shiftali al posto "giusto": bit 8 e 9
        MOVE.W  D1,D3           ; Salva in d3
;
        AND.W   #%111100,d2     ; Selez. i "vecchi" 4 bit dello scroll ad 1
                                ; pixel, max 16 pixel.
        LSR.W   #2,d2           ; Shiftali al posto giusto: primi 4 bits!
        OR.W    d2,d3           ; Salva in d3
;
        AND.W   #%11000000,d4   ; Selez. i bit alti: scatti di 16/32 pixel
        LSL.W   #4,d4           ; Posto giusto: BITS 10&11 per PF1
        OR.W    D4,d3           ; Salva in d3

        MOVE.w  D3,(A1)+        ; Salva il valore BPLCON1 finale
        CMP.L   a0,a2           ; Fine della tabella?
        BNE.S   CONVLOOP        ; Se non ancora, continua la conversione!
        RTS

; Tabella con i valori finali per il $dff102 (BPLCON1)

CON1VALUES:
        DCB.W   NUMVAL*2,0
CON1TABEND:

*****************************************************************************
; Routine che copia i valori dalla tabella CON1VALUES al bplcon1 in copper.
; Una volta letta tutta la tabella, smette.
*****************************************************************************

WABBLE:
        tst.w   FLAGGY                  ; Abbiamo finito la tabella?
        beq.s   NOWA                    ; Se si, esci!
        move.l  Con1TabPointer(PC),a0   ; Con1TabPointer in a0
        move.w  (a0)+,SCRLVAL           ; Copia il valore in copperlist
        cmp.l   #CON1TABEND,a0          ; Siamo alla fine della tab?
        bne.s   okay                    ; Se non ancora, ok
        clr.w   FLAGGY                  ; Altrimenti segna che abbiamo finito
okay:
        lea     Con1TabPointer(PC),a0   ; Con1TabPointer in a0
        addq.l  #2,(a0)                 ; Vai al prossimo valore
NOWA:
        RTS

FLAGGY:
        dc.w    -1

Con1TabPointer:
        dc.l    CON1VALUES
```

```
*****************************************************************************
;                       COPPERLIST AGA
*****************************************************************************

        CNOP    0,8

                Section MiaCop,data_C

AGACOPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop

        dc.w    $102            ; BplCon1
SCRLVAL:
        dc.w    0               ; Val. Bplcon1 - cambiato dalla routine

        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod
        dc.w    $10a,-8         ; Bpl2Mod

        dc.w    $1fc,3          ; Burst mode 64bit - NOTA: I bit alti del
                                ; BPLCON1 che permettono lo scroll a scatti
                                ; di 16 o 32 pixel funzionano solo se il
                                ; burst e' a 32 o 64 bit, rispettivamente.

EVENBPLPT:
        dc.w $e0,0,$e2,0                ;bitplane   0

                    ; 5432109876543210
        dc.w    $100,%0001001000000001  ; 1 bitplane LOWRES 320x256.

        dc.w    $106,$C00       ; Nibble alti
        dc.w    $180,$001       ; COLOR 0 REGISTER
        dc.w    $182,$081       ; COLOR 1 REGISTER
        dc.w    $106,$200       ; Nibble bassi
        dc.w    $180,$124       ; COLOR 0 REGISTER
        dc.w    $182,$567       ; COLOR 1 REGISTER

        dc.w    $FFFF,$FFFE     ; FineCopperist

*****************************************************************************
;                           BITPLANES
*****************************************************************************

        CNOP    0,8

PIC1:
        dcb.b   40*256,%00000111

        END
```

Lezione15g2

```
; Lezione15g2.s         - Scroll AGA dei bitplanes a scatti di 1/4 di pixel,
;                         per un messimo di 64 pixel. I 2 playfield scrollano
;                         indipendentemente in questo esempio.
```

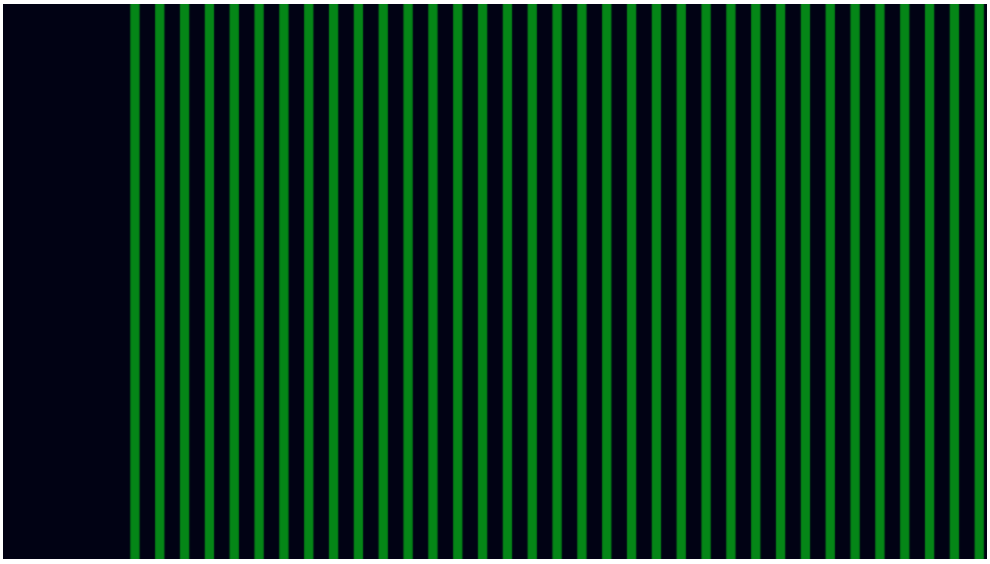


Figura 29.7: Lezione 15g1

```
; NOTA: I 2 bit alti dello scroll, che permettono "scatti" di 16 o di 32 pixel,
; per un massimo di 64 pixel di scroll, funzionano solo se il burst mode e'
; a 64 pixel (settando i 2 bit bassi di FMODE, ossia $dff1fc).

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA (planes pari)

        MOVE.L  #PIC1,d0
        LEA     EVENBPLPT,A1            ; BPL POINTERS DEI BITPLANES PARI
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)

;       Puntiamo la pic AGA (planes dispari)

        MOVE.L  #PIC2,d0
        LEA     ODDBPLPT,A1     ; BPL POINTERS DEI BITPLANES DISPARI
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
```

```
        bsr.w   MAKEMOVTAB      ; Questa semplice routine fa 2 tabelle
                                ; con valori da 0 a 255, poi di nuovo a 0

        bsr.w   FINESCROLLC     ; Questa routine "converte" i valori decimali
                                ; in valori di scroll per il BPLCON1 AGA

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #AgaCopList,$80(a5)     ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        BSR.w   WABBLE

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

*****************************************************************************
; Questa routine fa una tabella con valori "decimali" 0-255-0, in due tabelle
; separate, una per il playfield1 e l'altra per il playfield 2.
*****************************************************************************

NUMVAL = 255    ; Il nuovo bplcon1 puo' andare da 0 a 255, sfruttiamolo!

makemovtab:
        LEA     MOVTAB(PC),A0   ; Tab valori playfield 1
        LEA     MOVTAB2(PC),A1  ; Tab valori playfield 2
        MOVEQ   #0,D0           ; MINIMO PLAYFIELD 1 : 0
        MOVE.L  #NUMVAL,d2      ; MINIMO PLAYFIELD 2 : numval
        MOVE.L  #NUMVAL,D1      ; MASSIMO : numval
AMTLOOP:
        MOVE.W  D0,(A0)+        ; Val playfield 1
        MOVE.W  D2,(A1)+        ; Val playfield 2
        ADDQ.L  #1,D0           ; aggiungi 1 al val. playfield 1
        SUBQ.L  #1,D2           ; togli 1 al val. playfield 2
        CMP.L   D1,D0           ; pf1=numval? (allora pf2=zero)
        BNE.S   AMTLOOP         ; se non ancora, continua a loopare.
AMTLOOP2:
        MOVE.W  D0,(A0)+        ; Val Pf1 - (Da numval a 0)
        MOVE.W  D2,(A1)+        ; Val Pf2 - lascia sempre zero... fermo!
        SUBQ.L  #1,D0           ; Subba val. Pf1
```

```
        BNE.S   AMTLOOP2        ; d0=zero? (flag Z) - se non ancora loopa!
        RTS

MOVTAB:
        DCB.W   NUMVAL*2,0      ; *2 perche' sono words
MOVTABEND:

MOVTAB2:
        DCB.W   NUMVAL*2,0      ; *2 perche' sono words
MOVTAB2END:

*****************************************************************************
; Routine che converte da numeri "decimali" a valori per il bplcon1 AGA.
; In pratica scompone il numero a 8 bit posizionando i sui bit secondo lo
; schema del bplcon1 aga:
;
;       15      64 PIXEL SCROLL PF2 (AGA)
;       14      64 PIXEL SCROLL PF2 (AGA)
;       13      FINE SCROLL PF2 (AGA SCROLL 35ns 1/4 of pixel)
;       12      FINE SCROLL PF2
;       11      64 PIXEL SCROLL PF1 (AGA)
;       10      64 PIXEL SCROLL PF1 (AGA)
;       09      FINE SCROLL PF1 (AGA SCROLL 35ns 1/4 of pixel)
;       08      FINE SCROLL PF1
;       07      PF2H3
;       06      PF2H2
;       05      PF2H1
;       04      PF2H0
;       03      PF1H3
;       02      PF1H2
;       01      PF1H1
;       00      PF1H0

*****************************************************************************

FINESCROLLC:
        LEA     MOVTAB2(PC),A0          ; Tab valori playfield 2
        LEA     MOVTAB(PC),A3           ; Tab valori playfield 1
        LEA     CON1VALUES(PC),A1       ; Tab destinazione per $DFF002
        LEA     MOVTAB2END(PC),a2       ; Fine della tabella
CONVLOOP:
        MOVEQ   #0,D1
        MOVE.W  (A0)+,D1        ; VALORE "DECIMALE" PF1 IN D1
        MOVE.W  (A3)+,D5        ; VALORE "DECIMALE" PF2 IN D5
        MOVE.W  D1,D2           ; COPIA VAL. 1 IN D2
        MOVE.W  d1,d4           ; COPIA VAL. 1 IN D4
        MOVE.W  D5,D6           ; COPIA VAL. 2 IN D6
;pf1
        AND.W   #%11,D1         ; Selez. bits 0-1 (SCROLL 1/4 e 1/2 pixel)
        LSL.W   #8,D1           ; Shiftali al posto "giusto": bit 8 e 9
        MOVE.W  D1,D3           ; Salva in d3
;pf2
        AND.W   #%11,D5         ; Selez. bits 0-1 (SCROLL 1/4 e 1/2 pixel)
        LSL.W   #8,D5           ; Shiftali al posto "giusto", in 2 passaggi:
        LSL.W   #4,D5           ; in totale shiftati di  12 bit: bit 12 e 13
        OR.W    D5,D3           ; Salva in d3
;pf1
        AND.W   #%111100,d2     ; Selez. i "vecchi" 4 bit dello scroll ad 1
                                ; pixel, max 16 pixel.
        LSR.W   #2,d2           ; Shiftali al posto giusto: primi 4 bits!
        OR.W    d2,d3           ; Salva in d3
;pf2
```

```
        MOVE.W  D6,D5
        AND.W   #%111100,d5     ; Selez. i "vecchi" 4 bit dello scroll ad 1
                                ; pixel, max 16 pixel.
        LSL.W   #2,d5           ; Shiftali al posto giusto: bits 4,5,6,7
        OR.W    d5,d3           ; Salva in d3
;pf1
        AND.W   #%11000000,d4   ; Selez. i bit alti: scatti di 16/32 pixel
        LSL.W   #4,d4           ; Posto giusto: BITS 10&11 per PF1
        OR.W    D4,d3           ; Salva in d3
;pf2
        AND.W   #%11000000,d6   ; Selez. i bit alti: scatti di 16/32 pixel
        LSL.W   #8,d6           ; Posto giusto: BITS 14&15 per PF2
        OR.W    d6,d3           ; add pf2 16 pixel scroll bits to d3

        MOVE.w  D3,(A1)+        ; Salva il valore BPLCON1 finale
        CMP.L   a0,a2           ; Fine della tabella?
        BNE.S   CONVLOOP        ; Se non ancora, continua la conversione!
        RTS

; Tabella con i valori finali per il $dff102 (BPLCON1)

CON1VALUES:
        DCB.W   NUMVAL*2,0
CON1TABEND:

*****************************************************************************
; Routine che copia i valori dalla tabella CON1VALUES al bplcon1 in copper.
; Una volta letta tutta la tabella, smette.
*****************************************************************************

WABBLE:
        tst.w   FLAGGY                  ; Abbiamo finito la tabella?
        beq.s   NOWA                    ; Se si, esci!
        move.l  Con1TabPointer(PC),a0   ; Con1TabPointer in a0
        move.w  (a0)+,SCRLVAL           ; Copia il valore in copperlist
        cmp.l   #CON1TABEND,a0          ; Siamo alla fine della tab?
        bne.s   okay                    ; Se non ancora, ok
        clr.w   FLAGGY                  ; Altrimenti segna che abbiamo finito
okay:
        lea     Con1TabPointer(PC),a0   ; Con1TabPointer in a0
        addq.l  #2,(a0)                 ; Vai al prossimo valore
NOWA:
        RTS

FLAGGY:
        dc.w    -1

Con1TabPointer:
        dc.l    CON1VALUES

*****************************************************************************
;                       COPPERLIST AGA
*****************************************************************************

        CNOP    0,8

                Section MiaCop,data_C

AGACOPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
```

```
        dc.w    $94,$00d0       ; DdfStop

        dc.w    $102            ; BplCon1
SCRLVAL:
        dc.w    0               ; Val. Bplcon1 - cambiato dalla routine

        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod
        dc.w    $10a,-8         ; Bpl2Mod

        dc.w    $1fc,3          ; Burst mode 64bit - NOTA: I bit alti del
                                ; BPLCON1 che permettono lo scroll a scatti
                                ; di 16 o 32 pixel funzionano solo se il
                                ; burst e' a 32 o 64 bit, rispettivamente.

EVENBPLPT:
        dc.w $e0,0,$e2,0                ;bitplane   0
ODDBPLPT:
        dc.w $e4,0,$e6,0                ;bitplane   1

                    ; 5432109876543210
        dc.w    $100,%0010001000000001  ; 2 bitplane LOWRES 320x256.

        dc.w    $106,$C00       ; Nibble alti
        dc.w    $180,$001       ; COLOR 0 REGISTER
        dc.w    $182,$081       ; COLOR 1 REGISTER
        dc.w    $184,$aa1       ; COLOR 2 REGISTER
        dc.w    $186,$aa1       ; COLOR 3 REGISTER
        dc.w    $106,$200       ; Nibble bassi
        dc.w    $180,$124       ; COLOR 0 REGISTER
        dc.w    $182,$567       ; COLOR 1 REGISTER
        dc.w    $184,$413       ; COLOR 2 REGISTER
        dc.w    $186,$a83       ; COLOR 3 REGISTER

        dc.w    $FFFF,$FFFE     ; FineCopperist

*****************************************************************************
;                           BITPLANES
*****************************************************************************

        CNOP    0,8

PIC1:
        dcb.b   40*256,%00000111
PIC2:
        dcb.b   40*256,%01110000

        END
```

Lezione15g3

```
; Lezione15g3.s -       Ondeggiamento con il bplcon1 AGA di una figura LORES.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
```

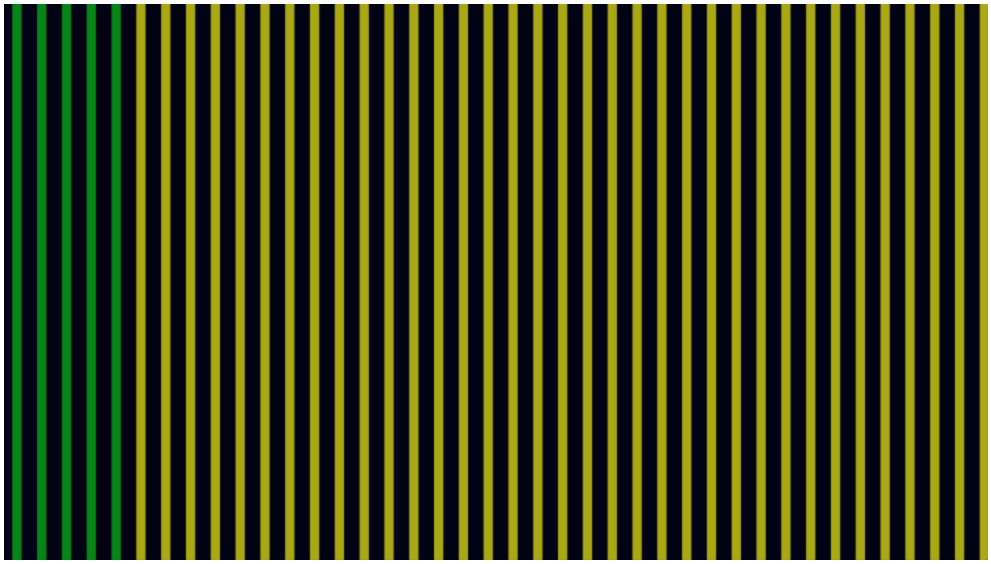


Figura 29.8: Lezione 15g2

```
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #40*256,d0      ; Lunghezza di un bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        bsr.w   FaiAgaCopCon1   ; Fai copperlist con WAIT+BPLCON2 ogni linea

        bsr.s   MettiColori     ; Metti i colori della pic

        bsr.w   FINESCROLLC2    ; Questa routine "converte" i valori decimali
                                ; in valori di scroll per il BPLCON1 AGA

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
```

```
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        BSR.w   WABBLE

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

;*****************************************************************************

MettiColori:
        LEA     PICTURE+(10240*8),A0    ; indirizzo della color palette alla
                                        ; fine della figura -> in A0
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi
```

```
; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Palette salvata in binario con il PicCon (opzioni: save as binary, non cop)

LogoPal:
        incbin  "Pic640x100x256.pal"

;******************************************************************************
; Routine che crea copperlist con un WAIT+BPLCON1 ogni linea
;******************************************************************************

FaiAgaCopCon1:
        lea     AgaCon1,a0      ; Indirizzo buffer in copperlist
        move.l  #$01020000,d0   ; BplCon1
        move.l  #$2c07fffe,d1   ; WAIT - Inizio linea Y=$2c
        move.w  #200-1,d7       ; Numero linee da fare
FaiAGALoopC:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        move.l  d0,(a0)+        ; BplCon1
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        dbra    d7,FaiAGALoopC
        rts

******************************************************************************
; Routine che converte da numeri "decimali" a valori per il bplcon1 AGA.
; In pratica scompone il numero a 8 bit posizionando i sui bit secondo lo
; schema del bplcon1 aga. Qesta versione da una sola tabella di valori 0-255
; converte in bplcon1, con lo stesso valore per i 2 playfield, adatto per
; scroll di figure come quella di questo esempio.
```

```
*****************************************************************************

FINESCROLLC2:
        LEA     MOVTAB(PC),A0           ; Tab valori
        LEA     CON1VALUES(PC),A1       ; Tab destinazione per $DFF002
        LEA     MOVTABEND(PC),a2        ; Fine della tabella
CONVLOOP:
        MOVEQ   #0,D1
        MOVEQ   #0,D2
        MOVEQ   #0,D3
        MOVEQ   #0,D4
        MOVE.B  (A0)+,D1        ; VALORE "DECIMALE" PF1 IN D1
        MOVE.W  D1,D2           ; COPIA VAL. 1 IN D2
        MOVE.W  d1,d4           ; COPIA VAL. 1 IN D4
;pf1
        AND.W   #%11,D1         ; Selez. bits 0-1 (SCROLL 1/4 e 1/2 pixel)
        LSL.W   #8,D1           ; Shiftali al posto "giusto": bit 8 e 9
        MOVE.W  D1,D3           ; Salva in d3
;pf2
        LSL.W   #4,D1           ; Shiftali al posto "giusto": bit 12 e 13
        OR.W    D1,D3           ; Salva in d3
;pf1
        AND.W   #%111100,d2     ; Selez. i "vecchi" 4 bit dello scroll ad 1
                                ; pixel, max 16 pixel.
        LSR.W   #2,d2           ; Shiftali al posto giusto: primi 4 bits!
        OR.W    d2,d3           ; Salva in d3
;pf2
        LSL.W   #4,d2           ; Shiftali al posto giusto: bits 4,5,6,7
        OR.W    d2,d3           ; Salva in d3
;pf1
        AND.W   #%11000000,d4   ; Selez. i bit alti: scatti di 16/32 pixel
        LSL.W   #4,d4           ; Posto giusto: BITS 10&11 per PF1
        OR.W    D4,d3           ; Salva in d3
;pf2
        LSL.W   #4,d4           ; Posto giusto: BITS 14&15 per PF2
        OR.W    d4,d3           ; add pf2 16 pixel scroll bits to d3

        MOVE.w  D3,(A1)+        ; Salva il valore BPLCON1 finale
        CMP.L   a0,a2           ; Fine della tabella?
        BNE.S   CONVLOOP        ; Se non ancora, continua la conversione!
        RTS



WABBLE:
        btst    #2,$dff016
        beq.s   wabble
        move.l  TabPointer(PC),a0       ; Punto attuale Tab in a0
        lea     Con1TabEnd(PC),a2       ; Fine Tab
        lea     AGACON1+6,a1            ; Effetto copper in a1
        move.w  #200-1,d7               ; numero linee, ossia loops
scroll:
        cmp.l   a2,a0           ; Fine tab?
        bne.s   okay            ; Se non ancora, continua
        LEA     CON1VALUES(PC),a0       ; Altrimenti riparti da capo.
okay:
        move.w  (a0)+,(a1)      ; Copia dalla tab al bplcon1 in copperlist
        addq.w  #8,a1           ; Salta il wait - al prossimo bplcon1
        dbra    d7,scroll       ; D0 VOLTE

        move.l  TabPointer(PC),a0       ; Punto attuale Tab in a0
        addq.w  #2,a0                   ; "scrollo" in avanti
```

```
        cmp.l   a2,a0           ; Fine tab?
        bne.s   okay2           ; Se non ancora, continua
        LEA     CON1VALUES(PC),a0       ; Altrimenti riparti da capo.
okay2:
        move.l  a0,TabPointer   ; Aggiorna il puntatore
        RTS

TabPointer:
        dc.l    CON1VALUES


; Tabella con i valori finali per il $dff102 (BPLCON1)

NUMVAL  EQU     400+300+200+100

CON1VALUES:
        DCB.W   NUMVAL,0
CON1TABEND:

;IS
;BEG>0
;END>360
;AMOUNT>400
;AMPLITUDE>127  ; Se la figura e' in LORES lo scroll va da 0 a 255!!
;YOFFSET>127

;AMOUNT>300

;AMOUNT>200

;AMOUNT>100

; Qua ci sono uno sotto l'altro 4 sintab...

MOVTAB:
        DC.B    $80,$82,$84,$86,$88,$8A,$8C,$8E,$90,$92,$94,$96,$98,$9A,$9C,$9E
        DC.B    $A0,$A1,$A3,$A5,$A7,$A9,$AB,$AD,$AF,$B1,$B2,$B4,$B6,$B8,$BA,$BB
        DC.B    $BD,$BF,$C1,$C2,$C4,$C6,$C7,$C9,$CA,$CC,$CE,$CF,$D1,$D2,$D4,$D5
        DC.B    $D7,$D8,$DA,$DB,$DC,$DE,$DF,$E0,$E1,$E3,$E4,$E5,$E6,$E7,$E9,$EA
        DC.B    $EB,$EC,$ED,$EE,$EF,$F0,$F1,$F1,$F2,$F3,$F4,$F5,$F5,$F6,$F7,$F7
        DC.B    $F8,$F9,$F9,$FA,$FA,$FB,$FB,$FC,$FC,$FC,$FD,$FD,$FD,$FD,$FE,$FE
        DC.B    $FE,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FD,$FD,$FD,$FD,$FC,$FC
        DC.B    $FC,$FB,$FB,$FA,$FA,$F9,$F9,$F8,$F7,$F7,$F6,$F5,$F5,$F4,$F3,$F2
        DC.B    $F1,$F1,$F0,$EF,$EE,$ED,$EC,$EB,$EA,$E9,$E7,$E6,$E5,$E4,$E3,$E1
        DC.B    $E0,$DF,$DE,$DC,$DB,$DA,$D8,$D7,$D5,$D4,$D2,$D1,$CF,$CE,$CC,$CA
        DC.B    $C9,$C7,$C6,$C4,$C2,$C1,$BF,$BD,$BB,$BA,$B8,$B6,$B4,$B2,$B1,$AF
        DC.B    $AD,$AB,$A9,$A7,$A5,$A3,$A1,$A0,$9E,$9C,$9A,$98,$96,$94,$92,$90
        DC.B    $8E,$8C,$8A,$88,$86,$84,$82,$80,$7E,$7C,$7A,$78,$76,$74,$72,$70
        DC.B    $6E,$6C,$6A,$68,$66,$64,$62,$60,$5E,$5D,$5B,$59,$57,$55,$53,$51
        DC.B    $4F,$4D,$4C,$4A,$48,$46,$44,$43,$41,$3F,$3D,$3C,$3A,$38,$37,$35
        DC.B    $34,$32,$30,$2F,$2D,$2C,$2A,$29,$27,$26,$24,$23,$22,$20,$1F,$1E
        DC.B    $1D,$1B,$1A,$19,$18,$17,$15,$14,$13,$12,$11,$10,$0F,$0E,$0D,$0D
        DC.B    $0C,$0B,$0A,$09,$09,$08,$07,$07,$06,$05,$05,$04,$04,$03,$03,$02
        DC.B    $02,$02,$01,$01,$01,$01,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00
        DC.B    $00,$00,$01,$01,$01,$01,$02,$02,$02,$03,$03,$04,$04,$05,$05,$06
        DC.B    $07,$07,$08,$09,$09,$0A,$0B,$0C,$0D,$0D,$0E,$0F,$10,$11,$12,$13
        DC.B    $14,$15,$17,$18,$19,$1A,$1B,$1D,$1E,$1F,$20,$22,$23,$24,$26,$27
        DC.B    $29,$2A,$2C,$2D,$2F,$30,$32,$34,$35,$37,$38,$3A,$3C,$3D,$3F,$41
        DC.B    $43,$44,$46,$48,$4A,$4C,$4D,$4F,$51,$53,$55,$57,$59,$5B,$5D,$5E
        DC.B    $60,$62,$64,$66,$68,$6A,$6C,$6E,$70,$72,$74,$76,$78,$7A,$7C,$7E

        DC.B    $80,$83,$86,$88,$8B,$8E,$90,$93,$95,$98,$9B,$9D,$A0,$A2,$A5,$A8
```

```
        DC.B    $AA,$AD,$AF,$B1,$B4,$B6,$B9,$BB,$BD,$C0,$C2,$C4,$C6,$C9,$CB,$CD
        DC.B    $CF,$D1,$D3,$D5,$D7,$D9,$DB,$DC,$DE,$E0,$E2,$E3,$E5,$E7,$E8,$EA
        DC.B    $EB,$EC,$EE,$EF,$F0,$F1,$F2,$F4,$F5,$F6,$F6,$F7,$F8,$F9,$FA,$FA
        DC.B    $FB,$FB,$FC,$FC,$FD,$FD,$FD,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FE,$FD
        DC.B    $FD,$FD,$FC,$FC,$FB,$FB,$FA,$FA,$F9,$F8,$F7,$F6,$F6,$F5,$F4,$F2
        DC.B    $F1,$F0,$EF,$EE,$EC,$EB,$EA,$E8,$E7,$E5,$E3,$E2,$E0,$DE,$DC,$DB
        DC.B    $D9,$D7,$D5,$D3,$D1,$CF,$CD,$CB,$C9,$C6,$C4,$C2,$C0,$BD,$BB,$B9
        DC.B    $B6,$B4,$B1,$AF,$AD,$AA,$A8,$A5,$A2,$A0,$9D,$9B,$98,$95,$93,$90
        DC.B    $8E,$8B,$88,$86,$83,$80,$7E,$7B,$78,$76,$73,$70,$6E,$6B,$69,$66
        DC.B    $63,$61,$5E,$5C,$59,$56,$54,$51,$4F,$4D,$4A,$48,$45,$43,$41,$3E
        DC.B    $3C,$3A,$38,$35,$33,$31,$2F,$2D,$2B,$29,$27,$25,$23,$22,$20,$1E
        DC.B    $1C,$1B,$19,$17,$16,$14,$13,$12,$10,$0F,$0E,$0D,$0C,$0A,$09,$08
        DC.B    $08,$07,$06,$05,$04,$04,$03,$03,$02,$02,$01,$01,$01,$00,$00,$00
        DC.B    $00,$00,$00,$00,$00,$01,$01,$01,$02,$02,$03,$03,$04,$04,$05,$06
        DC.B    $07,$08,$08,$09,$0A,$0C,$0D,$0E,$0F,$10,$12,$13,$14,$16,$17,$19
        DC.B    $1B,$1C,$1E,$20,$22,$23,$25,$27,$29,$2B,$2D,$2F,$31,$33,$35,$38
        DC.B    $3A,$3C,$3E,$41,$43,$45,$48,$4A,$4D,$4F,$51,$54,$56,$59,$5C,$5E
        DC.B    $61,$63,$66,$69,$6B,$6E,$70,$73,$76,$78,$7B,$7E

        DC.B    $81,$85,$89,$8D,$91,$95,$99,$9D,$A1,$A4,$A8,$AC,$B0,$B3,$B7,$BA
        DC.B    $BE,$C1,$C5,$C8,$CB,$CE,$D1,$D4,$D7,$DA,$DD,$E0,$E2,$E5,$E7,$E9
        DC.B    $EB,$ED,$EF,$F1,$F3,$F4,$F6,$F7,$F8,$F9,$FA,$FB,$FC,$FD,$FD,$FE
        DC.B    $FE,$FE,$FE,$FE,$FE,$FD,$FD,$FC,$FB,$FA,$F9,$F8,$F7,$F6,$F4,$F3
        DC.B    $F1,$EF,$ED,$EB,$E9,$E7,$E5,$E2,$E0,$DD,$DA,$D7,$D4,$D1,$CE,$CB
        DC.B    $C8,$C5,$C1,$BE,$BA,$B7,$B3,$B0,$AC,$A8,$A4,$A1,$9D,$99,$95,$91
        DC.B    $8D,$89,$85,$81,$7D,$79,$75,$71,$6D,$69,$65,$61,$5D,$5A,$56,$52
        DC.B    $4E,$4B,$47,$44,$40,$3D,$39,$36,$33,$30,$2D,$2A,$27,$24,$21,$1E
        DC.B    $1C,$19,$17,$15,$13,$11,$0F,$0D,$0B,$0A,$08,$07,$06,$05,$04,$03
        DC.B    $02,$01,$01,$00,$00,$00,$00,$00,$00,$01,$01,$02,$03,$04,$05,$06
        DC.B    $07,$08,$0A,$0B,$0D,$0F,$11,$13,$15,$17,$19,$1C,$1E,$21,$24,$27
        DC.B    $2A,$2D,$30,$33,$36,$39,$3D,$40,$44,$47,$4B,$4E,$52,$56,$5A,$5D
        DC.B    $61,$65,$69,$6D,$71,$75,$79,$7D

        DC.B    $83,$8B,$93,$9B,$A2,$AA,$B1,$B9,$C0,$C6,$CD,$D3,$D9,$DE,$E3,$E8
        DC.B    $EC,$F0,$F4,$F6,$F9,$FB,$FC,$FD,$FE,$FE,$FD,$FC,$FB,$F9,$F6,$F4
        DC.B    $F0,$EC,$E8,$E3,$DE,$D9,$D3,$CD,$C6,$C0,$B9,$B1,$AA,$A2,$9B,$93
        DC.B    $8B,$83,$7B,$73,$6B,$63,$5C,$54,$4D,$45,$3E,$38,$31,$2B,$25,$20
        DC.B    $1B,$16,$12,$0E,$0A,$08,$05,$03,$02,$01,$00,$00,$01,$02,$03,$05
        DC.B    $08,$0A,$0E,$12,$16,$1B,$20,$25,$2B,$31,$38,$3E,$45,$4D,$54,$5C
        DC.B    $63,$6B,$73,$7B
MOVTABEND:

;***************************************************************************
;*                              COPPERLIST                                 *
;***************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

; Nota: il ddfstart/stop HIRES sarebbero $003c e $00d4, ma con il burst attivo
; va bene lo stesso valore del LOWRES, ossia $0038 e $00d0.

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
```

```
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%0000001000010001  ; 8 bitplane Lores 640x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,3          ; Burst mode a 64 bit

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
```

```
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI
```

```
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

AgaCon1:
        dcb.l   200*2           ; Ossia: 200 linee * 2 long:
                                ; 1 per il wait,
                                ; 1 per il bplcon1

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

        end

La striscina che sale ogni tanto sulla sinistra e' un mistero.
Dato che la routine funziona, credo sia un bug dell'hardware dell'Amiga!
```

Lezione15g4

```
; Lezione15g4.s -       Ondeggiamento con il bplcon1 AGA di una figura HIRES.
;                       Da notare che se la pic e' HIRES lo scroll puo'
;                       andare da 0 a 127, per un totale di 32 pixel lowres.
;                       In pratica non e' abilitato il bit piu' alto.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
```

```
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #80*100,d0      ; Lunghezza di un bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        bsr.w   FaiAgaCopCon1   ; Fai copperlist con WAIT+BPLCON2 ogni linea

        bsr.s   MettiColori     ; Metti i colori della pic

        bsr.w   FINESCROLLC2    ; Questa routine "converte" i valori decimali
                                ; in valori di scroll per il BPLCON1 AGA

        lea     $dff000,a5
        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

        BSR.w   WABBLE

        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

;******************************************************************************

MettiColori:
        LEA     LogoPal(PC),A0          ; indirizzo della color palette
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco
```

```
DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Palette salvata in binario con il PicCon (opzioni: save as binary, non cop)

LogoPal:
        incbin  "Pic640x100x256.pal"

;*****************************************************************************
; Routine che crea copperlist con un WAIT+BPLCON1 ogni linea
;*****************************************************************************
```

```
FaiAgaCopCon1:
        lea     AgaCon1,a0      ; Indirizzo buffer in copperlist
        move.l  #$01020000,d0   ; BplCon1
        move.l  #$2c07fffe,d1   ; WAIT - Inizio linea Y=$2c
        move.w  #99-1,d7        ; Numero linee da fare
FaiAGALoopC:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        move.l  d0,(a0)+        ; BplCon1
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        dbra    d7,FaiAGALoopC
        rts

*****************************************************************************
; Routine che converte da numeri "decimali" a valori per il bplcon1 AGA.
; In pratica scompone il numero a 8 bit posizionando i sui bit secondo lo
; schema del bplcon1 aga. Qesta versione da una sola tabella di valori 0-255
; converte in bplcon1, con lo stesso valore per i 2 playfield, adatto per
; scroll di figure come quella di questo esempio.
*****************************************************************************

FINESCROLLC2:
        LEA     MOVTAB(PC),A0           ; Tab valori
        LEA     CON1VALUES(PC),A1       ; Tab destinazione per $DFF002
        LEA     MOVTABEND(PC),a2        ; Fine della tabella
CONVLOOP:
        MOVEQ   #0,D1
        MOVEQ   #0,D2
        MOVEQ   #0,D3
        MOVEQ   #0,D4
        MOVE.B  (A0)+,D1        ; VALORE "DECIMALE" PF1 IN D1
        MOVE.W  D1,D2           ; COPIA VAL. 1 IN D2
        MOVE.W  d1,d4           ; COPIA VAL. 1 IN D4
;pf1
        AND.W   #%11,D1         ; Selez. bits 0-1 (SCROLL 1/4 e 1/2 pixel)
        LSL.W   #8,D1           ; Shiftali al posto "giusto": bit 8 e 9
        MOVE.W  D1,D3           ; Salva in d3
;pf2
        LSL.W   #4,D1           ; Shiftali al posto "giusto": bit 12 e 13
        OR.W    D1,D3           ; Salva in d3
;pf1
        AND.W   #%111100,d2     ; Selez. i "vecchi" 4 bit dello scroll ad 1
                                ; pixel, max 16 pixel.
        LSR.W   #2,d2           ; Shiftali al posto giusto: primi 4 bits!
        OR.W    d2,d3           ; Salva in d3
;pf2
        LSL.W   #4,d2           ; Shiftali al posto giusto: bits 4,5,6,7
        OR.W    d2,d3           ; Salva in d3
;pf1
        AND.W   #%11000000,d4   ; Selez. i bit alti: scatti di 16/32 pixel
        LSL.W   #4,d4           ; Posto giusto: BITS 10&11 per PF1
        OR.W    D4,d3           ; Salva in d3
;pf2
        LSL.W   #4,d4           ; Posto giusto: BITS 14&15 per PF2
        OR.W    d4,d3           ; add pf2 16 pixel scroll bits to d3

        MOVE.w  D3,(A1)+        ; Salva il valore BPLCON1 finale
        CMP.L   a0,a2           ; Fine della tabella?
        BNE.S   CONVLOOP        ; Se non ancora, continua la conversione!
        RTS
```

```
WABBLE:
        move.l  TabPointer(PC),a0       ; Punto attuale Tab in a0
        lea     Con1TabEnd(PC),a2       ; Fine Tab
        lea     AGACON1+6,a1            ; Effetto copper in a1
        move.w  #99-1,d7                ; numero linee, ossia loops
scroll:
        move.w  (a0)+,(a1)      ; Copia dalla tab al bplcon1 in copperlist
        addq.w  #8,a1           ; Salta il wait - al prossimo bplcon1
        cmp.l   a2,a0           ; Fine tab?
        bne.s   okay            ; Se non ancora, continua
        LEA     CON1VALUES(PC),a0       ; Altrimenti riparti da capo.
okay:
        dbra    d7,scroll       ; D0 VOLTE

        move.l  TabPointer(PC),a0       ; Punto attuale Tab in a0
        addq.w  #2,a0                   ; "scrollo" in avanti
        cmp.l   a2,a0           ; Fine tab?
        bne.s   okay2           ; Se non ancora, continua
        LEA     CON1VALUEs(PC),a0       ; Altrimenti riparti da capo.
okay2:
        move.l  a0,TabPointer   ; Aggiorna il puntatore
        RTS

TabPointer:
        dc.l    CON1VALUES


; Tabella con i valori finali per il $dff102 (BPLCON1)

NUMVAL  EQU     256


CON1VALUES:
        DCB.W   NUMVAL,0
CON1TABEND:

        ds.b    10000

;IS
;BEG>0
;END>360
;AMOUNT>128
;AMPLITUDE>63   ; Se la figura e' in HIRES lo scroll va da 0 a 127!!
;YOFFSET>63

;AMOUNT>64
;AMPLITUDE>63   ; Se la figura e' in HIRES lo scroll va da 0 a 127!!
;YOFFSET>63

;AMOUNT>32
;AMPLITUDE>63   ; Se la figura e' in HIRES lo scroll va da 0 a 127!!
;YOFFSET>63

;AMOUNT>32
;AMPLITUDE>63   ; Se la figura e' in HIRES lo scroll va da 0 a 127!!
;YOFFSET>63


MOVTAB:
        DC.B    $41,$44,$47,$4A,$4D,$50,$53,$56,$59,$5B,$5E,$61,$63,$66,$68,$6A
        DC.B    $6D,$6F,$71,$73,$74,$76,$77,$79,$7A,$7B,$7C,$7C,$7D,$7E,$7E,$7E
```

```
        DC.B    $7E,$7E,$7E,$7D,$7C,$7C,$7B,$7A,$79,$77,$76,$74,$73,$71,$6F,$6D
        DC.B    $6A,$68,$66,$63,$61,$5E,$5B,$59,$56,$53,$50,$4D,$4A,$47,$44,$41
        DC.B    $3D,$3A,$37,$34,$31,$2E,$2B,$28,$25,$23,$20,$1D,$1B,$18,$16,$14
        DC.B    $11,$0F,$0D,$0B,$0A,$08,$07,$05,$04,$03,$02,$02,$01,$00,$00,$00
        DC.B    $00,$00,$00,$01,$02,$02,$03,$04,$05,$07,$08,$0A,$0B,$0D,$0F,$11
        DC.B    $14,$16,$18,$1B,$1D,$20,$23,$25,$28,$2B,$2E,$31,$34,$37,$3A,$3D

        DC.B    $42,$48,$4E,$54,$5A,$5F,$65,$69,$6E,$72,$75,$78,$7A,$7C,$7D,$7E
        DC.B    $7E,$7D,$7C,$7A,$78,$75,$72,$6E,$69,$65,$5F,$5A,$54,$4E,$48,$42
        DC.B    $3C,$36,$30,$2A,$24,$1F,$19,$15,$10,$0C,$09,$06,$04,$02,$01,$00
        DC.B    $00,$01,$02,$04,$06,$09,$0C,$10,$15,$19,$1F,$24,$2A,$30,$36,$3C

        DC.B    $45,$51,$5D,$67,$70,$77,$7B,$7E,$7E,$7B,$77,$70,$67,$5D,$51,$45
        DC.B    $39,$2D,$21,$17,$0E,$07,$03,$00,$00,$03,$07,$0E,$17,$21,$2D,$39

        DC.B    $45,$51,$5D,$67,$70,$77,$7B,$7E,$7E,$7B,$77,$70,$67,$5D,$51,$45
        DC.B    $39,$2D,$21,$17,$0E,$07,$03,$00,$00,$03,$07,$0E,$17,$21,$2D,$39
MOVTABEND:


;***************************************************************************
;*                              COPPERLIST                                 *
;***************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop

; Nota: il ddfstart/stop HIRES sarebbero $003c e $00d4, ma con il burst attivo
; va bene lo stesso valore del LOWRES, ossia $0038 e $00d0.

        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; Bpl1Mod (burst 64bit, modulo=modulo-8)
        dc.w    $10a,-8         ; Bpl2Mod (come sopra)

                ; 5432109876543210
        dc.w    $100,%1000001000010001  ; 8 bitplane HIRES 640x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,3          ; Burst mode a 64 bit

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
```

```
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
```

```
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

AgaCon1:
        dcb.l   99*2            ; Ossia: 99 linee * 2 long:
                                ; 1 per il wait,
                                ; 1 per il bplcon1
        dc.w    $9007,$fffe     ; aspetta la fine del logo
        dc.w    $100,$200       ; zero bitplanes

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;******************************************************************************
```

```
; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "Pic640x100x256.RAW"    ; (C) by Cristiano "KREEX" Evangelisti

        end

Per le figure hires abbiamo una limitazione hardware nello scroll: non e'
abilitato il bit piu' alto, quello dello scroll di 32 pixel alla volta,
per cui il valore va da 0 a 127, anziche' da 0 a 255.
```

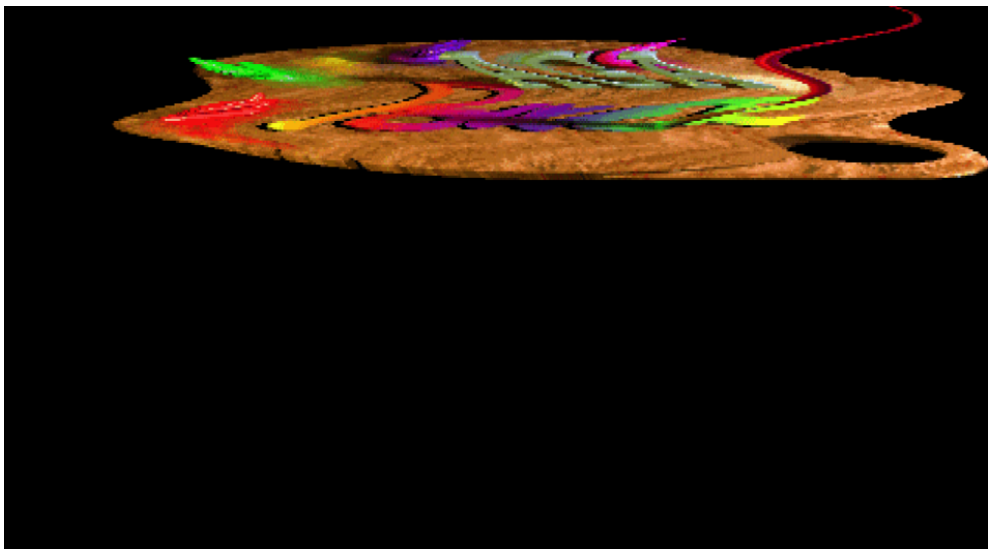


Figura 29.9: Lezione 15g4

29.8 Lezione15h

```
; Lezione15h.s          - Esempio degli effetti del BPLCON4 nella palette

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"

*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:
```

```
;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7                 ;num of bitplanes -1
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #10240,d0               ;lenght of bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        bsr.s   MettiColori

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

LOOP:
        MOVE.L  #$1ff00,d1      ; bit per la selezione tramite AND
        MOVE.L  #$11000,d2      ; linea da aspettare = $110
Waity1:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BNE.S   Waity1

Aspetta:
        MOVE.L  4(A5),D0        ; VPOSR e VHPOSR - $dff004/$dff006
        ANDI.L  D1,D0           ; Seleziona solo i bit della pos. verticale
        CMPI.L  D2,D0           ; aspetta la linea $110
        BEQ.S   Aspetta

        bsr.s   CambiaCon4

        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

CambiaCon4:
        addq.b  #1,ContaSpetta
        cmp.b   #20,ContaSpetta ; Passati 20 frames?
        bne.s   NonAncora
        clr.b   ContaSpetta     ; resetta il "timer"
        addq.b  #1,MioCon4      ; Cambia il byte alto di bplcon4
NonAncora:
        rts

ContaSpetta:
        dc.w    0

MettiColori:
        LEA     PICTURE+(10240*8),A0    ; indirizzo della color palette alla
                                        ; fine della figura -> in A0
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
```

```
        LEA     COLPOB+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
                                ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.
```

```
;*****************************************************************************
;*                              COPPERLIST                                   *
;*****************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$2c81       ; DiwStrt
        dc.w    $90,$2cc1       ; DiwStop
        dc.w    $92,$0038       ; DdfStart
        dc.w    $94,$00d0       ; DdfStop
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,0          ; Bpl1Mod
        dc.w    $10a,0          ; Bpl2Mod
        dc.w    $10c            ; BplCon4
MioCon4:
        dc.w    0

                ; 5432109876543210
        dc.w    $100,%0000001000010001  ; 8 bitplane LOWRES 320x256. Per
                                        ; settare 8 planes setto il bit 4 e
                                        ; azzero i bit 12,13,14. Il bit 0 e'
                                        ; settato dato che abilita molte
                                        ; funzioni AGA che vedremo dopo.

        dc.w    $1fc,0          ; Burst mode azzerato (per ora!)

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLP0B:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
```

```
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI
```

```
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "MURALE320*256*256c.RAW"

        end
```

Questo esempio non e' certo "bello", ma rende l'idea del funzionamento del registro, che e' certamente utile in certe routines che potreste fare, appositamente studiate per cambiare la palette.... potreste per esempio mettere un $10c ogni linea per cambiare la palette in modo diverso molte volte... per fare chissacche'....

29.9 Lezione15i

```
; Lezione15i.s  - Visualiziamo la prima pic in 640x480 32Khz VGA non lace.
;                 Se non avete un monitor adatto vedete solo dei disturbi.

        SECTION AgaRulez,CODE

;       Include "DaWorkBench.s" ; togliere il ; prima di salvare con "WO"
```

```
*****************************************************************************
        include "startup2.s"    ; Salva Copperlist Etc.
*****************************************************************************

                ;5432109876543210
DMASET  EQU     %1000001110000000       ; copper, bitplane DMA

WaitDisk        EQU     30      ; 50-150 al salvataggio (secondo i casi)

START:

;       Puntiamo la pic AGA

        MOVE.L  #PICTURE,d0
        LEA     BPLPOINTERS,A1
        MOVEQ   #8-1,D7         ; num. bitplanes
POINTB:
        move.w  d0,6(a1)
        swap    d0
        move.w  d0,2(a1)
        swap    d0
        addi.l  #80*100,d0      ; Lunghezza di un bitplane
        addq.w  #8,a1
        dbra    d7,POINTB               ;Rifai D1 volte (D1=num of bitplanes)

        move.l  #$2c07fffe,d1   ; Prima linea YY wait: $2c
        moveq   #$00,d5         ; Colore start
        move.w  #99-1,d7        ; Numero linee: 99
        bsr.w   FaiAGACopB      ; Fai una sfumatura BLU

        bsr.s   MettiColori

        MOVE.W  #DMASET,$96(a5)         ; DMACON - abilita bitplane, copper
        move.l  #CopList,$80(a5)        ; Puntiamo la nostra COP
        move.w  d0,$88(a5)              ; Facciamo partire la COP
        move.w  #0,$1fc(a5)             ; Fmode azzerato, burst normale
        move.w  #$c00,$106(a5)          ; BPLCON3 resettato
        move.w  #$11,$10c(a5)           ; BPLCON4 resettato

                 ;5432109876543210
        MOVE.W  #%0001101110001000,$1DC(A5) ; BEACON0 - lista dei bit settati:

                        ; 3 - BLANKEN - COMPOSITE BLANK OUT TO CSY PIN
                        ; 7 - VARBEAMEN - VARIABLE BEAM COUNTER COMP. ENABLED
                        ;     Abilita i comparatori variabili di beam per
                        ;     operare nel contatore orizzontare principale,
                        ;     e disabilita lo stop hardware del display in
                        ;     orizzontale e in verticale.
                        ; 8 - VARHSYEN - VARIABLE HORIZONTAL SYNC ENABLED
                        ;     Attiva i registri HSSTRT/HSSTOP (var. HSY)
                        ; 9 - VARVSYEN - VARIABLE VERTICAL SYNC ENABLED
                        ;     Attiva i registri VSSTRT/VSSTOP (var. VSY)
                        ; 11- LOLDIS - DISABLE LONGLINE/SHORTLINE TOGGLE
                        ;     Disabilita lo scambio tra linee lunghe/corte.
                        ; 12- VARVBEN - VARIABLE VERTICAL BLANK ENABLED
                        ;     Attiva i registri VBSTRT/VBSTOP, e disabilita la
                        ;     "fine" hardware della finestra video.

        MOVE.W  #113,$1C0(a5)   ; HTOTAL - HIGHEST NUMBER COUNT, HORIZ LINE
                                ; Color clock massimo per linea orizzontale:
                                ; Il VGA ha 114 colorclocks per scan line!
```

```
                            ; Il valore va da 0 a 255: 113 va bene!

        MOVE.W  #%1000,$1C4(a5) ; HBSTRT - HORIZONTAL LINE POS FOR HBLANK START
                            ; I bit 0-7 contengono le posizioni di start
                            ; e di stop del blanking orizzontale in
                            ; incrementi di 280ns. I bit 8-10 servono per
                            ; un posizionamento a 35ns (1/4 di pixel).
                            ; In questo caso abbiamo settato 2240ns.

        MOVE.W  #14,$1DE(a5)    ; HORIZONTAL SYNC START - Numero di color
                            ; clocks per il Sync-start.

        MOVE.W  #28,$1C2(a5)    ; HORIZONTAL LINE POSITION FOR HSYNC STOP
                            ; Num. di color-clocks per Sync-stop.

        MOVE.W  #30,$1C6(a5)    ; HORIZONTAL LINE POSITION FOR HBLANK STOP
                            ; Linea orizzontale di stop Horiz BLANK

        MOVE.W  #70,$1E2(a5)    ; HCENTER - POS. ORIZZ. di VSYNCH in interlace
                            ; nel caso di beam counters variabili.

        MOVE.W  #524,$1C8(a5)   ; VTOTAL - HIGHEST NUMBERED VERTICAL LINE
                            ; Massima linea verticale numerata, ossia
                            ; la linea alla quale resettare il contatore
                            ; diposizione verticale.
                            ; Sappiamo che il modo VGA ha 525 linee.

        MOVE.W  #0,$1CC(a5)     ; VBSTRT - VERTICAL LINE FOR VBLANK START
        MOVE.W  #3,$1E0(a5)     ; VERTICAL SYNC START

        MOVE.W  #5,$1CA(a5)     ; VERTICAL LINE POSITION FOR VSYNC STOP
        MOVE.W  #29,$1CE(a5)    ; VBSTOP - VERTICAL LINE FOR VBLANK STOP

        MOVE.W  #%0000110000100001,$106(a5)     ; 0 - external blank enable
                                                ; 5 - BORDER BLANK
                                                ; 10-11 AGA dual playfiled fix
LOOP:
        BTST    #6,$BFE001
        BNE.S   LOOP
        RTS

*****************************************************************************

MettiColori:
        LEA     LogoPal(PC),A0          ; indirizzo della color palette
        LEA     COLP0+2,A1              ; Indirizzo del primo registro
                                        ; settato per i nibble ALTI
        LEA     COLP0B+2,A2             ; Indirizzo del primo registro
                                        ; settato per i nibble BASSI
        MOVEQ   #8-1,d7                 ; 8 banchi da 32 registri ciascuno
ConvertiPaletteBank:
        moveq   #0,d0
        moveq   #0,d2
        moveq   #0,d3
        moveq   #32-1,d6        ; 32 registri colore per banco

DaLongARegistri:        ; loop che trasforma i colori $00RrGgBb.l nelle 2
                        ; word $0RGB, $0rgb adatte ai registri copper.

; Conversione dei nibble bassi da $00RgGgBb (long) al colore aga $0rgb (word)

        MOVE.B  1(A0),(a2)      ; Byte alto del colore $00Rr0000 copiato
```

```
                            ; nel registro cop per nibble bassi
        ANDI.B  #%00001111,(a2) ; Seleziona solo il nibble BASSO ($0r)
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        lsl.b   #4,d2           ; Sposta a sinistra di 4 bit il nibble basso
                                ; del GREEN, "trasformandolo" in nibble alto
                                ; di del byte basso di D2 ($g0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24bit
        ANDI.B  #%00001111,d3   ; Seleziona solo il nibble BASSO ($0b)
        or.b    d2,d3           ; "FONDI" i nibble bassi di green e blu...
        move.b  d3,1(a2)        ; Formando il byte basso finale $gb da mettere
                                ; nel registro colore, dopo il byte $0r, per
                                ; formare la word $0rgb dei nibble bassi

; Conversione dei nibble alti da $00RgGgBb (long) al colore aga $0RGB (word)

        MOVE.B  1(A0),d0        ; Byte alto del colore $00Rr0000 in d0
        ANDI.B  #%11110000,d0   ; Seleziona solo il nibble ALTO ($R0)
        lsr.b   #4,d0           ; Shifta a destra di 4 bit il nibble, in modo
                                ; che diventi il nibble basso del byte ($0R)
        move.b  d0,(a1)         ; Copia il byte alto $0R nel color register
        move.b  2(a0),d2        ; Prendi il byte $0000Gg00 dal colore a 24bit
        ANDI.B  #%11110000,d2   ; Seleziona solo il nibble ALTO ($G0)
        move.b  3(a0),d3        ; Prendi il byte $000000Bb dal colore a 24 bit
        ANDI.B  #%11110000,d3   ; Seleziona solo il nibble ALTO ($B0)
        lsr.b   #4,d3           ; Shiftalo di 4 bit a destra trasformandolo in
                                ; nibble basso del byte basso di d3 ($0B)
        ori.b   d2,d3           ; Fondi i nibble alti di green e blu ($G0+$0B)
        move.b  d3,1(a1)        ; Formando il byte basso finale $GB da mettere
                                ; nel registro colore, dopo il byte $0R, per
                                ; formare la word $0RGB dei nibble alti.

        addq.w  #4,a0           ; Saltiamo al prossimo colore .l della palette
                                ; attaccata in fondo alla pic
        addq.w  #4,a1           ; Saltiamo al prossimo registro colore per i
                                ; nibble ALTI in Copperlist
        addq.w  #4,a2           ; Saltiamo al prossimo registro colore per i
                                ; nibble BASSI in Copperlist

        dbra    d6,DaLongARegistri

        add.w   #(128+8),a1     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble ALTI
        add.w   #(128+8),a2     ; salta i registri colore + il dc.w $106,xxx
                                ; dei nibble BASSI

        dbra    d7,ConvertiPaletteBank  ; Converte un banco da 32 colori per
        rts                             ; loop. 8 loop per i 256 colori.

; Palette salvata in binario con il PicCon (opzioni: save as binary, non cop)

LogoPal:
        incbin  "Pic640x100x256.pal"

;******************************************************************************
; Routine che crea sfumature aga BLU:
;
; d1 = Prima linea da aspettare (Wait, ad es: $2c07fffe per linea Y=$2c)
; d5 = inizio tonalita' ($00-$ff)
; d7 = Numero linee da fare
;******************************************************************************

FaiAgaCopB:
```

```
        lea     AgaCopEff1,a0
        move.l  #$01060c00,d4   ; BplCon3 - nibble alti
        move.l  #$01060e00,d3   ; BplCon3 - nibble bassi
        move.w  #$180,d2        ; Registro Color0
FaiAGALoopB:
        move.l  d1,(a0)+        ; Metti il wait YYXXFFFE
        add.l   #$01000000,d1   ; Fai waitare una linea sotto per la prossima
        move.l  d4,(a0)+        ; BplCon3 - selez. nibble alti
        move.w  d2,(a0)+        ; Registro Color0
        addq.b  #1,d5           ; "Illumina" leggermente il colore $Gg
        move.w  d5,d6           ; Copialo in d6
        and.w   #%11110000,d6   ; Selez. solo il nibble ALTO
        lsr.w   #4,d6           ; Alla posizione giusta, ossia al BLU $xxB)
        move.w  d6,(a0)+        ; Valore del Color0 (nib alti)
        move.l  d3,(a0)+        ; BplCon3 - selez. nibble bassi
        move.w  d2,(a0)+        ; Registro Color0
        move.w  d5,d6           ; Colore $xx in d6
        and.w   #%00001111,d6   ; Selez. solo i nibble bassi - posizione $xxB
        move.w  d6,(a0)+        ; Metti il colore in copperlist (nibble bassi)
        dbra    d7,FaiAGALoopB
        rts

;******************************************************************************
;*                              COPPERLIST                                    *
;******************************************************************************

        CNOP    0,8     ; Allineo a 64 bit

        section coppera,data_C

COPLIST:
        dc.w    $8E,$1c45       ; diwstrt VGA
        dc.w    $90,$ffe5       ; diwstop VGA
        dc.w    $92,$0018       ; ddfstrt VGA
        dc.w    $94,$0068       ; ddfstop VGA
        dc.w    $1e4,$100
        dc.w    $102,0          ; BplCon1
        dc.w    $104,0          ; BplCon2
        dc.w    $108,-8         ; modulo (-8 per burst 64 bit)
        dc.w    $10A,-8         ; -8


                ; 5432109876543210
        dc.w    $100,%0000001001010001  ; 8 bitplane SHIRES 640x480 VGA.

        dc.w    $1fc,$8003      ; sprite scan doubling???

BPLPOINTERS:
        dc.w $e0,0,$e2,0        ; primo         bitplane
        dc.w $e4,0,$e6,0        ; secondo          "
        dc.w $e8,0,$ea,0        ; terzo            "
        dc.w $ec,0,$ee,0        ; quarto           "
        dc.w $f0,0,$f2,0        ; quinto           "
        dc.w $f4,0,$f6,0        ; sesto            "
        dc.w $f8,0,$fA,0        ; settimo          "
        dc.w $fC,0,$fE,0        ; ottavo           "

; In questo caso la palette viene aggiornata da una routine, per cui basta
; lasciare azzerati i valori dei registri.

        DC.W    $106,$c00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE ALTI
COLP0:
```

```
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$e00       ; SELEZIONA PALETTE 0 (0-31), NIBBLE BASSI
COLPOB:
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2C00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$2E00      ; SELEZIONA PALETTE 1 (32-63), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4C00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$4E00      ; SELEZIONA PALETTE 2 (64-95), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6C00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$6E00      ; SELEZIONA PALETTE 3 (96-127), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8C00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$8E00      ; SELEZIONA PALETTE 4 (128-159), NIBBLE BASSI
```

```
        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AC00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$AE00      ; SELEZIONA PALETTE 5 (160-191), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CC00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$CE00      ; SELEZIONA PALETTE 6 (192-223), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EC00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE ALTI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        DC.W    $106,$EE00      ; SELEZIONA PALETTE 7 (224-255), NIBBLE BASSI

        DC.W    $180,0,$182,0,$184,0,$186,0,$188,0,$18A,0,$18C,0,$18E,0
        DC.W    $190,0,$192,0,$194,0,$196,0,$198,0,$19A,0,$19C,0,$19E,0
        DC.W    $1A0,0,$1A2,0,$1A4,0,$1A6,0,$1A8,0,$1AA,0,$1AC,0,$1AE,0
        DC.W    $1B0,0,$1B2,0,$1B4,0,$1B6,0,$1B8,0,$1BA,0,$1BC,0,$1BE,0

        dc.w    $106,%0000110000100001          ; 0 - external blank enable
                                                ; 5 - BORDER BLANK
                                                ; 10-11 AGA dual playfiled fix

AgaCopEff1:
        dcb.l   99*5            ; Ossia: 99 linee * 5 long:
                                ; 1 per il wait,
                                ; 1 per il bplcon3
                                ; 1 per color0 (nib alti)
                                ; 1 per il bplcon3
                                ; 1 per color0 (nib bassi)
        dc.w    $9007,$fffe     ; aspetta la fine del logo
        dc.w    $100,$201       ; zero bitplanes
```

```
        dc.w    $FFFF,$FFFE     ; Fine della copperlist

;*****************************************************************************

; Figura RAW ad 8 bitplanes, cioe' a 256 colori

        CNOP    0,8     ; allineo a 64 bit

PICTURE:
        INCBIN  "Pic640x100x256.RAW"    ; (C) by Cristiano "KREEX" Evangelisti

        end
```

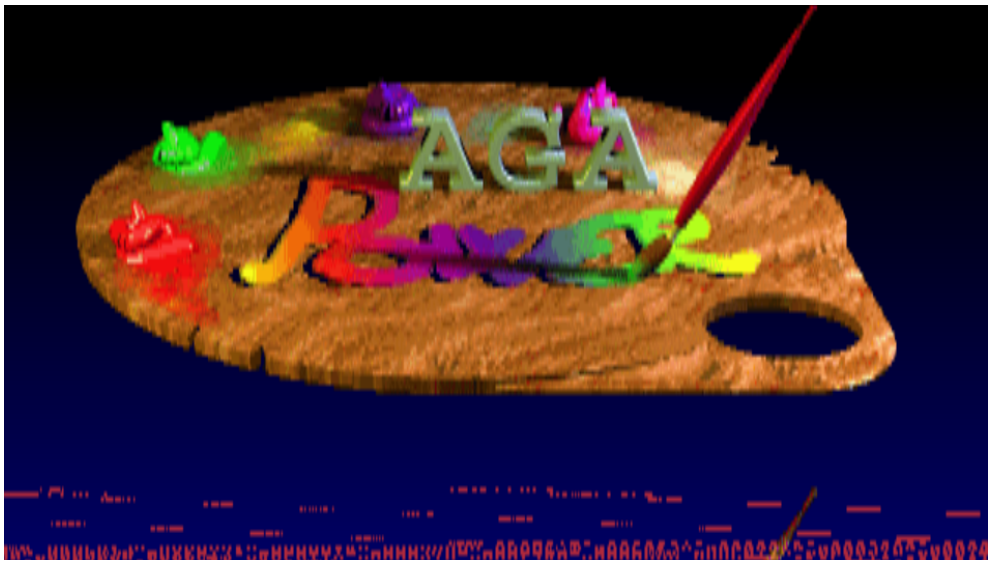


Figura 29.10: Lezione 15i

Parte III

Approfondimenti

CAPITOLO 30

TEXTURE MAPPING

Autore: Alberto Longo

30.1 Premessa

Questo articolo ha lo scopo (e la pretesa) di spiegare nella maniera più semplice possibile i principi alla base del texture mapping in real time, con particolare riferimento ad alcune delle tecniche da me utilizzate nella realizzazione del motore del videogioco BREATHLESS. Esso è stato scritto nel poco tempo lasciatomi libero dalla programmazione di Breathless, dagli impegni di lavoro e dalla mia ragazza, per cui non può e non deve essere considerato come una fonte inesauribile di conoscenza, ma solo come un ottimo punto di partenza per un argomento così affascinante ed attuale. Ciò nonostante, posso assicurare senza dubbio alcuno che i lettori di questo articolo risparmieranno una notevole quantità di notti insonni, notti che io stesso ho passato nel disperato tentativo di capire come hanno fatto quelli della Id software a realizzare quel capolavoro che è Doom.

Data la complessità dell'argomento si presuppone una certa esperienza nella programmazione in assembly e, comunque, un'abbondante dose di buona volontà. Il mio consiglio è quello di leggere più volte l'articolo, nonché i sorgenti e la documentazione allegata. Per ogni approfondimento, relativo anche al 3D in generale, rimando a libri ed articoli specifici.

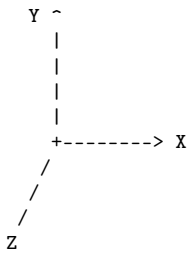
Per una buona lettura di questo articolo non è possibile prescindere da una buona conoscenza della struttura di Amiga e dell'assembly 68000+. Si suppone quindi che il lettore abbia una certa dimestichezza con tali argomenti, essendo impossibile, per ovvi motivi, soffermarsi su di essi.

L'articolo è volutamente scritto in maniera molto semplice, cercando di evitare disquisizioni troppo complesse o troppo vicine al rigido formalismo matematico, questo per permettere una agevole lettura al maggior numero possibile di persone.

Se non diversamente specificato, tutti i riferimenti riguardanti la parte hardware sono relativi alle macchine AGA.

Gli esempi di codice riportati, sono scritti in pseudo-linguaggio, oppure in assembly, e hanno scopo esclusivamente didattico. Questo significa che non sono ottimizzati nel migliore dei modi e che non sono stati testati, per cui la presenza di errori non è esclusa.

Gli assi del sistema di riferimento nello spazio sono orientati come segue:



30.2 Cenni sul formato dei numeri in virgola fissa

Nell'elaborazione di oggetti tridimensionali si ha spessissimo a che fare con numeri non interi. Tale tipo di dato è comunemente implementato nei linguaggi ad alto livello come il C e il BASIC, tramite il formato in virgola mobile. Con tale formato è possibile rappresentare un ampio insieme di numeri decimali, perdendo precisione solo quando strettamente necessario.

In Assembly le cose stanno in maniera decisamente diversa e, a meno che non si abbia a disposizione una FPU, è assolutamente improponibile l'utilizzo del formato in virgola mobile dei numeri, perché anche una semplice addizione dovrebbe essere realizzata da una routine di una certa complessità.

Si ricorre allora al formato in virgola fissa con il quale è possibile rappresentare numeri decimali utilizzando i normali numeri interi e quindi i registri del 68000. Risulta però necessario decidere in anticipo quanti bit si vogliono dedicare alla parte intera e quanti alla parte frazionaria e ciò si traduce in una scarsa flessibilità di questo tipo di rappresentazione. Il nostro scopo è però quello di eseguire il più velocemente possibile le operazioni sui numeri decimali, per cui bisogna sottostare a qualche compromesso.

Indicando che un numero è nel formato a virgola fissa, bisogna quindi specificare di quanti bit è composta la parte intera e di quanti bit è composta la parte frazionaria. In genere si utilizza la notazione x.y, dove x è il numero di bit dedicati alla parte intera e y il numero di bit per la parte frazionaria. Quindi, ad esempio, scrivendo 24.8 si vuole indicare che il numero è composto da 24 bit per la parte intera e da 8 bit per la parte frazionaria. In genere il formato più utilizzato è 16.16 (16 bit di parte intera e 16 di parte frazionaria), ed è quello a cui verrà fatto riferimento qui di seguito. Per ovvi motivi è conveniente che la somma del numero di bit dedicati alle due parti sia pari al numero di bit della più lunga parola che il processore è in grado di trattare, nel nostro caso 32.

La conversione da un numero decimale ad un numero in virgola fissa e viceversa si effettua tenendo conto della semplice formuletta:

```
virgola_fissa = INT(decimale * 2^bit_parte_frazionaria)
```

Ad esempio il numero decimale 12.3456, convertito nel formato 16.16 è pari a 12.3456*65536=809081 (la parte frazionaria ovviamente si perde), mentre all'inverso si avrà 809081/65536=12.34559631. Come si può notare c'è una perdita di precisione che è tanto più contenuta quanto più alto è il numero di bit dedicati alla parte frazionaria.

La somma di due numeri in virgola fissa si effettua senza nessun particolare accorgimento rispetto ai numeri interi. L'istruzione

```
1    add.l  d0,d1
```

è tutto quello che serve per sommare due numeri in virgola fissa contenuti rispettivamente in d0 e d1.

Il discorso è un pò più complesso per quel che riguarda le divisioni e le moltiplicazioni. Mi si conceda l'approssimazione contenuta nella seguente affermazione: un numero decimale A, nella sua forma in virgola fissa è pari ad A*K, dove K vale 2^bit_parte_frazionaria. Il prodotto dei due numeri decimali A e B nel formato in virgola fissa, vale:

```
A*K * B*K = (A * B) * K^2
```

Per ottenere il risultato cercato (A*B*K), si rende quindi necessaria una divisione per K (o meglio, uno shift a destra di 16 bit).

Analogamente, la divisione di due numeri A e B, vale:

```
A*K / B*K = A / B
```

Per evitare che la parte frazionaria venga annullata è sufficiente moltiplicare il dividendo per K:

```
A*K*K / B*K = (A / B) * K
```

I microprocessori dal 68020 in poi sono particolarmente versatili rispetto all'implementazione dei numeri in virgola fissa, in quanto dotati di istruzioni di moltiplicazione e divisione a precisione estesa. Non bisogna però dimenticare che tali istruzioni sono comunque più lente di quelle normali, per cui in certi casi può essere preferibile utilizzare numeri in virgola fissa che entrino in una word piuttosto che in una long word.

30.3 Cos'è il Texture Mapping

Il texture mapping è una tecnica per "attaccare" un'immagine grafica in bitmap (brush) o un'immagine calcolata matematicamente (texture algoritmica) ai poligoni o, più in generale, ad una qualunque entità tridimensionale. La normale grafica vettoriale appare alquanto spoglia ed irreale, in quanto ogni oggetto è composto da un insieme di poligoni, ognuno dei quali è riempito con un unico colore. Il texture mapping aggiunge ai semplici poligoni un maggiore realismo ed una maggiore profondità, consentendo di realizzare ambienti virtuali molto più verosomiglianti e, quindi, più spettacolari.

Abbiamo quindi a disposizione un'immagine bidimensionale di dimensioni note (la nostra texture), a cui possiamo accedere tramite coordinate (u,v) per conoscere il colore di un punto. Per ovvi motivi la texture è conservata in memoria in formato chunky pixel (ogni pixel corrisponde ad un byte), ovvero come una matrice di byte. Volendo mappare la texture su un poligono nello spazio, dobbiamo trovare un sistema che ci permetta di associare ad ogni punto (x,y,z) del poligono nello spazio, un punto (u,v) della texture. Il poligono nello spazio appartiene ad un piano, a cui associamo un sistema di riferimento bidimensionale, definito da un'origine e da due versori. Se il poligono è un rettangolo, è sufficiente scegliere come origine il primo vertice e calcolare le componenti dei versori a partire dai due lati che hanno in comune il primo vertice.

Detti quindi:

```
P(x,y,z) il generico punto del poligono di cui vogliamo calcolare il
         colore;

T(u,v)   il punto della texture associato a P;

O        l'origine del sistema di riferimento del poligono;

i, j     i versori del sistema di riferimento del poligono;

*        il simbolo di prodotto scalare;
```

possiamo scrivere:

```
T = ((P-O)*i,(P-O)*j)
```

Conosciamo ora le coordinate del punto T(u,v) che possiamo usare per leggere dalla texture il colore del punto P.

Come si può notare i calcoli da effettuare per ogni punto sono troppo complessi per un'applicazione in real-time. Come è possibile semplificare e velocizzare il tutto ?

Un primo importante passo è la semplificazione del problema. Lo scopo che ci prefiggiamo è quello di realizzare un motore che ci permetta di "passeggiare" all'interno di un mondo tridimensionale, e per fare ciò è sufficiente avere la possibilità di muoversi e di ruotare lo sguardo a destra o a sinistra. Questo porta ad alcune semplificazioni che renderanno il motore notevolmente più veloce:

1. muri e pavimenti devono essere tra loro perpendicolari
2. è possibile variare la posizione dell'osservatore solo sugli assi X e Z e non sull'asse Y
3. è possibile ruotare lo sguardo solo intorno all'asse Y

Questo significa che il nostro mondo è in effetti bidimensionale. Il nostro intento è quello di farlo sembrare tridimensionale.

30.4 Come viene implementato il texture mapping in applicazioni real-time?

Un primo semplice esempio

Supponiamo di voler utilizzare come texture un brush dalle dimensioni di 128x128 pixel e di volerlo mappare su un poligono di forma quadrata che, una volta ruotato, traslato e proiettato in 2d, appare a video come un quadrato dalle dimensioni di 64x64 pixel. Banalmente, bisognerà tracciare sul quadrato a video solo un pixel ogni 2 (2=128/64). Se invece il quadrato a video ha dimensioni di 32x32 pixel, basta tracciare un pixel ogni 4 (4=128/32). E' quindi facile intuire l'importanza della semplice relazione:

```
Step = BrushDim / ScreenDim
```

dove:

```
Step      : Passo (si tratta di un numero con virgola)
BrushDim  : Dimensione iniziale del brush
ScreenDim : Dimensione a video del brush
```

Volendo infatti mappare la texture su un quadrato a video, di lato pari a ScreenDim, potremmo scrivere qualcosa del tipo:

```
Step = BrushDim / ScreenDim
for y=0 to ScreenDim
    v = y * Step;
    for x=0 to ScreenDim
        u = x * Step
        WriteScreenPixel(x,y,ReadTexturePixel(u,v))
    endfor
endfor
```

dove:

```
Step, u, v              sono variabili in floating point

ReadTexturePixel(u,v)   è la funzione che legge  il colore del pixel
                        di coordinate (u,v) della texture;

WriteScreenPixel(x,y,c) è la funzione che scrive  un pixel di colore
                        c a video, alle coordinate (x,y).
```

Ma quello che ci interessa è la velocità, e questa routine è ancora decisamente lenta. C'è prima di tutto bisogno di un accesso diretto alla memoria, e di eliminare dai cicli le istruzioni lente (le moltiplicazioni):

```
Step = BrushDim / ScreenDim
v = 0
for y=0 to ScreenDim
    u = 0
    screen = ScreenBase + 320 * y
    for x=0 to ScreenDim
        screen[x] = texture[v][u]
        u += Step
    endfor
    v += Step
endfor
```

dove:

```
ScreenBase    è l'indirizzo di uno schermo in chunky pixel;
```

Come si può notare, le moltiplicazioni sono state sostituite da somme, mentre per la lettura di pixel dalla texture e per la scrittura a video sono stati utilizzati accessi diretti alla memoria, tramite gli array screen[] e texture[][]. Inoltre sia la texture che lo schermo sono organizzati in chunky pixel.

Per fare di meglio è più conveniente passare all'assembly:

```
1  ;a0 = ptr alla texture
2  ;a1 = ptr allo schermo chunky
3  ;d0 = u (nel formato 16.16, cioè 16 bit interi e 16 bit frazionari)
4  ;d1 = v (nel formato 16.16)
5  ;d2 = offset all'interno della texture
6  ;d4 = Step (nel formato 16.16)
7  ;d5 = ScreenDim * 320
8  ;d6 = x
9  ;d7 = y
10
11         moveq    #0,d1            ;v=0
12         move.w   ScreenDim,d5
13         mulu.w   #320,d5          ;Sapete come si ottimizza questo, no ?
14         moveq    #0,d7            ;inizializza x
15 loopy   moveq    #0,d0            ;u=0
16         move.l   ScreenBase,a0
17         add.l    d7,a0            ;a0=ptr alla riga attuale a schermo
18         move.l   d1,d2
19         clr.w    d2
20         swap     d2
21         lsl.w    #8,d2            ;d2=offset riga attuale della texture
22         move.w   ScreenDim,d6
23         subq.w   #1,d6            ;inizializza y
24 loopx   swap     d0
25         move.b   d0,d2
26         swap     d0
27         move.b   (a1,d2.l),(a0)+  ;Copia pixel da texture a schermo
28         add.l    d4,d0            ;u+=Step
```

```
29          dbra    d6,loopx
30          add.l   d4,d1           ;v+=Step
31          add.l   #320,d7
32          cmp.l   d5,d7
33          bne     loopy
```

Come si può facilmente intuire, l'esempio qui riportato non effettua altro che lo zoom di un brush, al variare di ScreenDim, ma è estremamente significativo per la comprensione dei principi che sono alla base del texture mapping e di una sua implementazione in applicazioni real-time.

È importante fare attenzione al fatto che il quadrato di cui sopra viene suddiviso in una serie di trattini orizzontali. Ogni trattino è a sua volta composto da un certo numero di pixel. A questo punto i pixel da tracciare a schermo vengono "campionati" dal brush ad una distanza gli uni dagli altri pari a Step.

Si osservi questo semplice esempio in cui il brush originario ha dimensioni di 10x10 pixel, mentre le dimensioni a video sono di 5x5. Il valore di Step è, ovviamente, 10 / 5 = 2 per cui, dal brush saranno scelti solo i trattini orizzontali di posto pari e, all'interno di ogni trattino, solo i pixel di posto pari:

```
    Brush 10x10               A video 5x5

A . B . C . D . E .
. . . . . . . . . .
F . G . H . I . J .             A B C D E
. . . . . . . . . .             F G H I J
K . L . M . N . O .     --->    K L M N O
. . . . . . . . . .             P Q R S T
P . Q . R . S . T .             U V W X Y
. . . . . . . . . .
U . V . W . X . Y .
. . . . . . . . . .
```

Se, invece, le dimensioni a video sono di 4x4, Step avrà valore pari a 10 / 4 = 2.5 e il risultato sarà:

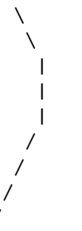
```
    Brush 10x10               A video 4x4

A . B . . C . D . .
. . . . . . . . . .
E . F . . G . H . .             A B C D
. . . . . . . . . .             E F G H
. . . . . . . . . .     --->    I J K L
I . J . . K . L . .             M N O P
. . . . . . . . . .
M . N . . O . P . .
. . . . . . . . . .
. . . . . . . . . .
```

Passiamo a qualcosa di più concreto

E' stato già detto che il mondo tridimensionale in cui abbiamo intenzione di muoverci ha solo muri verticali e l'unica rotazione permessa è quella intorno all'asse Y. Ogni muro è banalmente rappresentato da un poligono; lo stesso dicasi per ogni pezzo di pavimento o soffitto.

Supponiamo quindi di dover mappare la texture su un quadrato ruotato intorno all'asse Y. Il quadrato appare a video come un trapezio ruotato di 90 gradi e rappresenta un muro:

```
|\
| \
```



Come si può facilmente notare, questa figura è formata da una serie di trattini verticali di lunghezza decrescente, ovvero composti da un numero di pixel decrescente. Per ogni trattino verticale è sufficiente eseguire un ciclo simile al seguente:

```
loop    move.b  (a0,d0.w),(a1)  ;Copia il pixel
        add.w   d3,d1           ;Somma la parte frazionaria
        addx.w  d2,d0           ;Somma la parte intera (+ riporto)
        adda.l  d4,a1           ;Sposta il ptr. allo schermo
        dbra    d7,loop
```

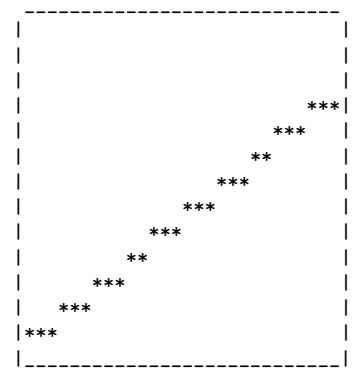
dove:

```
d0 = parte intera del contatore
d1 = parte frazionaria del contatore
d2 = parte intera dello Step
d3 = parte frazionaria dello Step
d4 = numero di pixel per ogni riga di shermo
d7 = numero di pixel da tracciare per il trattino corrente
a0 = ptr. alla colonna della texture corrispondente al trattino
a1 = ptr. al pixel corrente nello schermo
```

La texture è conservata in memoria come una matrice organizzata per colonne, questo per semplificare l'accesso ad ogni pixel della colonna corrente della texture. Il calcolo della colonna della texture da visualizzare, va fatto tenendo conto delle leggi della prospettiva.

Pavimenti e soffitti

Nel tipo di motore 3D che si intende realizzare, pavimenti e soffitti sono perfettamente orizzontali, nonchè perpendicolari ai muri. Il texture mapping di poligoni di questo genere è un pò più complesso di quello dei muri in quanto è necessario scorrere la texture secondo linee oblique. Inoltre è necessario effettuare il tracciamento per strisce orizzontali di pixel e non verticali, come avviene per i muri. Si osservi la seguente figura, rappresentante la texture (da 64x64 pixel) da mappare su un pezzo di pavimento (o soffitto):



Per ogni striscia orizzontale di pavimento (o soffitto) è necessario scorrere la texture secondo una linea non necessariamente orizzontale o verticale. Tale linea è rappresentata in figura per mezzo di asterischi (*). Il loop di texture mapping è qualcosa di simile al seguente:

```
for x = x1 to x2
    WriteScreenPixel(x,y,ReadTexturePixel(u & 63, v & 63))
    u += du
    v += dv
endfor
```

dove:

```
x1   = colonna iniziale della striscia orizzontale di pixel
x2   = colonna finale della striscia orizzontale di pixel
y    = riga su cui si trova la striscia di pixel
u, v = coordinate all'interno della texture
du   = valore di somma per u
dv   = valore di somma per v
```

Il problema, a questo punto, è costituito dal calcolo dei valori iniziali di u, v, du, dv. La tecnica adottata per il calcolo di tali valori dipende dall'approccio utilizzato nella realizzazione del motore.

Per chiarire un pò meglio le idee, si tenga presente che il poligono da tracciare appartiene al pavimento e, quindi, ad un piano. Su tale piano sono "incollate", una vicino all'altra, le texture da 64x64 pixel, in modo da coprirlo per intero. Ogni punto di tale piano è individuato tramite la coppia di coordinate (u,v) e, seguendo le regole della prospettiva, corrisponde ad un pixel a video di coordinate (x,y).

Le coordinate a video del punto iniziale e del punto finale della striscia orizzontale di pixel sono noti e sono rispettivamente (x1,y) e (x2,y). A tali punti corrispondono i punti (u1,v1) e (u2,v2) nel piano di texture che vanno calcolati. Il valore iniziale della coppia (u,v) è proprio (u1,v1), mentre il valore di (du,dv) è dato da:

```
du=(u2-u1)/(x2-x1+1)
dv=(v2-v1)/(x2-x1+1)
```

A titolo di esempio, consiglio di dare un'occhiata al sorgente AMOS presente nel file "TMap-Floor.lha".

30.5 Il calcolo della scena da tracciare

Riguardo al texture mapping sono molto utilizzati i due termini "ray-casting" e "BSP", ma non tutti sanno a cosa esattamente si riferiscano. Per tracciare a video una scena, non c'è solo bisogno di sapere come tracciarla, ma anche e soprattutto di sapere cosa tracciare. Il ray-casting e i BSP tree sono due dei metodi più diffusi per calcolare cosa tracciare in base al punto di vista dell'osservatore. In un classico labirinto alla Wolfestein è contenuto un numero molto elevato di poligoni, e volerli analizzare tutti per decidere quali fanno effettivamente parte della scena da visualizzare, è assurdo. C'è bisogno di tecniche più veloci e sia il ray-casting che i BSP, ci vengono in aiuto.

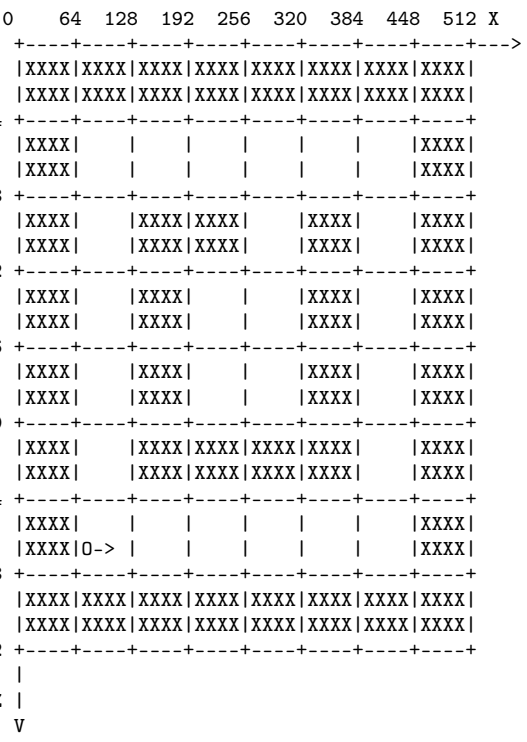
Ray-Casting

Viene spontaneo notare che il ray-casting ha un nome simile al più famoso ray-tracing (l'algoritmo utilizzato per immagini 3D fotorealistiche), ed in effetti la somiglianza non si ferma al nome. L'algoritmo dell ray-tracing consiste nel tracciare un raggio (una linea) tra l'osservatore ed ognuno dei pixel che compongono lo schermo. Per ognuno di questi raggi vengono poi calcolate collisioni e rifrazioni in modo da ricavare il colore del pixel corrispondente.

Il ray-casting non è altro che una semplificazione del ray-tracing: viene tracciato un solo raggio per ogni colonna dello schermo. Il guadagno in termini di velocità risulta subito evidente se si pensa che per calcolare un frame a 320x200 pixel sono necessari 320 raggi all'algoritmo del ray-casting contro i 64000 del ray-tracing !

Quella che potrebbe sembrare una esagerata approssimazione è invece un'idea tanto semplice quanto geniale. Non bisogna infatti dimenticare che il mondo che intendiamo visualizzare è soggetto a dei limiti per cui pavimenti e pareti sono tra loro perpendicolari. In più è possibile muoversi solo su un piano (la coordinata Y non può variare).

Si osservi il grafico seguente:



Esso rappresenta una semplice mappa bidimensionale organizzata per blocchi che possono essere pieni o vuoti. Un blocco pieno non può essere attraversato, un blocco vuoto si. Ogni blocco deve essere immaginato come una specie di cubo di dimensioni fisse, in genere 64 unità (o pixel) per lato, la cui faccia inferiore poggia sulla mappa, nella posizione indicata dalla griglia. La faccia inferiore e quella superiore rappresentano rispettivamente pavimento e soffitto dei blocchi vuoti. Le quattro facce perpendicolari al pavimento rappresentano altrettanti pezzi di muro. Ad ogni faccia è associata una texture di dimensioni 64x64 pixel.

Un blocco ha, quindi, sei parametri: un puntatore alla texture del pavimento, un puntatore alla texture del soffitto e quattro puntatori alle texture delle rimanenti quattro facce. Se il blocco

è pieno, non ha bisogno dei puntatori alle texture di soffitto e pavimento (sono posti a zero), mentre se è vuoto, non ha bisogno dei puntatori alle texture dei quattro pezzi di muro.

L'osservatore ha tre parametri: coordinata x, coordinata z e angolo di osservazione. Nel grafico precedente l'osservatore è rappresentato da una O e da una freccia che indica la direzione dello sguardo. Dividendo le coordinate x e z per la dimensione dei blocchi si calcola facilmente su quale blocco si trova l'osservatore.

Si immagini ora lo schermo come composto da 320 colonne da 200 pixel ciascuna.

L'algoritmo del ray-casting si può riassumere nei seguenti passi:

1. Tenendo conto dell'angolo di osservazione, calcolare il raggio (la retta) da tracciare tra l'osservatore e ognuna delle 320 colonne che compongono lo schermo.
2. A partire dal blocco su cui si trova l'osservatore e usando l'algoritmo di tracciamento delle linee di Bresenham, esaminare ogni blocco attraverso cui passa il raggio corrente finché non se ne trova uno pieno.
3. Se il blocco è pieno, vuol dire che c'è un'intersezione tra il raggio e due dei 4 lati del blocco. Calcolare le coordinate del punto di intersezione del lato più vicino all'osservatore. In questo modo, grazie ad una semplice operazione di and, è anche possibile calcolare quale dei 64 pixel del lato è stato colpito dal raggio. Questo date è estremamente utile durante il texture mapping vero e proprio.
4. Calcolare la distanza tra l'osservatore ed il punto di intersezione.
5. Calcolare l'altezza del muro in pixel (tenendo conto delle leggi della prospettiva) nel punto di intersezione.
6. Tracciare a schermo, nella colonna corrente, tramite una routine di texture mapping, il pezzo di muro intersecato.
7. Tornare al punto 2 finché non sono stati tracciati i 320 raggi.

L'algoritmo, come si può notare, è molto semplice, ma implementarlo in maniera efficace è tutt'altro discorso. A tal proposito consiglio di studiare i sorgenti e, soprattutto, il file di documentazione `notes.txt` contenuti nell'archivio `ack3d.zip` allegato a questo articolo. Tale archivio contiene, oltre ai sorgenti, un'analisi approfondita di una delle possibili implementazioni dell'algoritmo del ray-casting. A quanti riuscissero ad implementare in un programma funzionante le soluzioni proposte dall'autore dell'archivio, consiglio di studiare nuove e più efficienti strade per raggiungere i medesimi risultati. Posso testimoniare in prima persona che si può fare molto di meglio.

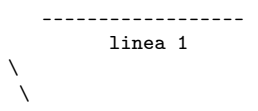
I sorgenti sono per PC, ma il loro valore didattico resta immutato.

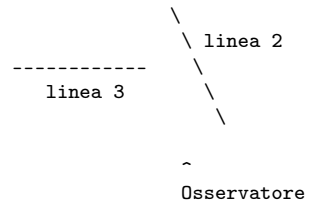
BSP Trees

Più complessa, invece, è l'idea di base dei BSP tree. Prima di tutto, BSP tree significa per esteso: Binary SPlit tree, ovvero alberi binari di divisione.

Vediamo come funzionano:

Si osservi la seguente figura, in cui 3 linee risiedono sul piano:





Volendo disegnare la scena utilizzando il classico algoritmo del pittore, bisognerebbe prima di tutto ordinare le linee in base alla distanza, quindi tracciarle in ordine dalla più lontana alla più vicina. Questa tecnica, oltre ad essere decisamente lenta, è anche soggetta a notevoli errore di imprecisione spesso difficili da eliminare.

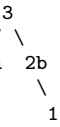
Utilizzando i BSP, il calcolo dell'ordine di tracciamento viene fatto una volta per tutte al di fuori del motore 3D, creando un albero binario contenente tutte le informazioni necessarie a tracciare nell'ordine giusto le linee, qualunque sia la posizione dell'osservatore.

Prima di tutto si noti che, dato un qualunque punto (x,y), si può sempre dire se esso si trova su un lato o sull'altro di una linea. Se il punto dovesse appartenere alla linea, lo si può considerare come appartenente ad uno dei due lati.

L'albero binario dei BSP è costituito da una serie di nodi che rappresentano le linee che si vogliono tracciare. Alla destra di ogni nodo si mettono tutte le linee che si trovano su un lato, mentre alla sinistra si mettono tutte le linee che si trovano sull'altro lato. Così, riguardo all'esempio precedente, l'albero potrebbe essere:

Ma è possibile utilizzare qualunque altra linea come nodo di testa:

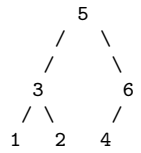
Volendo però utilizzare la linea 3 come nodo di testa, sorge un problema: su quale lato della linea 3 si trova la linea 2 ? La risposta è semplice: su entrambi. Si procede allora a suddividere la linea 2 in due parti, tagliate a metà dal prolungamento della linea 3. L'albero assume, quindi, questo aspetto:



Le linee 2a e 2b sono porzioni dell'originaria linea 2. Tracciando entrambe le linee a video, si otterrà esattamente la linea 2.

Durante la creazione di un albero BSP è necessario cercare di minimizzare il numero di suddivisioni di linee, pena un aumento spropositatto delle dimensioni dell'albero stesso, e quindi del tempo necessario al tracciamento di una scena.

Per tracciare una scena bisogna partire dal nodo di testa e calcolare su quale lato della linea è posto l'osservatore. Si visita il nodo relativo all'altro lato, si traccia la linea corrente, e poi si visita il nodo relativo al lato su cui si trova l'osservatore, il tutto in maniera ricorsiva. Ad esempio, il seguente albero:



genera la seguente sequenza se l'osservatore si trova a destra di tutte le linee:

```
4 - 6 - 5 - 2 - 3 - 1
```

L'estensione di questi concetti al 3D è semplice. Si considerino al posto delle linee, dei poligoni e si supponga che il poligono 1 sia il nodo di testa. Per sapere dove inserire il poligono 2 nell'albero è sufficiente calcolare su quale lato si trovano tutti i suoi punti rispetto al poligono 1. Se una parte del poligono 2 dovesse essere su un lato e l'altra parte sull'altro lato del poligono 1, si dovrebbe dividere il poligono 2 in due parti. Per fare ciò è sufficiente prendere in considerazione la linea formata dall'intersezione tra il poligono 2 e il piano a cui appartiene il poligono 1 e suddividere il poligono 2 lungo questa linea.

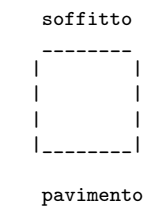
E' però da notare che per realizzare un motore simile a Doom, è sufficiente utilizzare i BSP tree nel caso bidimensionale. Il motivo risulterà chiaro dopo aver letto il paragrafo successivo.

Come si può facilmente comprendere, i BSP tree hanno notevoli vantaggi sul ray-casting: sono più veloci, danno la possibilità di tracciare muri obliqui e di ogni dimensione e, più in generale, danno la possibilità di realizzare ambienti più complessi e realistici. Per contro c'è da indicare un maggiore difficoltà d'uso

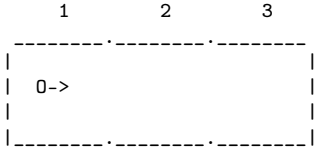
30.6 Saliamo le scale!

Le tecniche descritte fino a questo punto permettono di visualizzare scene tratte da un mondo sostanzialmente bidimensionale. Non si tratta cioè di vero 3D, ma solo di una parvenza.

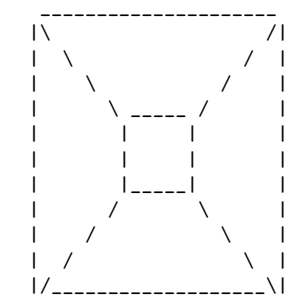
Un passo avanti nella realizzazione di un mondo tridimensionale più realistico può essere fatto con una tecnica abbastanza semplice. Si prenda in considerazione un blocco vuoto dell'algoritmo del ray-casting descritto poco prima. Se l'osservatore fosse nel blocco, ovviamente, si troverebbe tra pavimento e soffitto. La sezione laterale del blocco si presenta così:



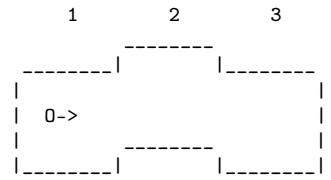
Sempre considerando una sezione laterale, si accostino 2 blocchi al precedente:



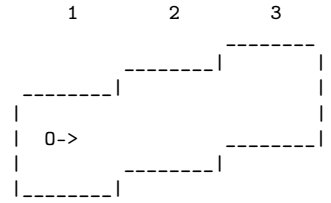
L'osservatore, che si trova sul blocco 1, vede (approssimativamente) questa scena (un corridoio):



I tre blocchi si trovano alla stessa altezza ma cosa succede se, ad esempio, il blocco centrale si trova più in alto rispetto agli altri due ?



L'osservatore non vede più solo un semplice corridoio, ma vede anche un gradino. Innalzando il blocco 3 rispetto al blocco 2 si avrà:



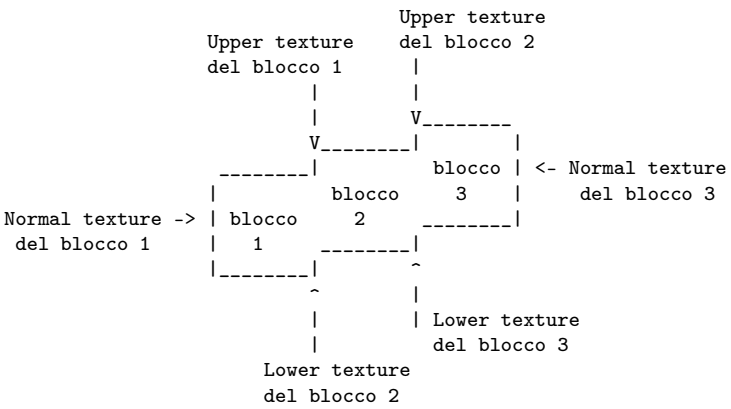
Per ognuna delle quattro facce laterali (quindi per ogni muro), sono ora definite tre texture:

Normal è la texture visualizzata tra soffitto e pavimento, se il blocco è pieno;

Upper è la texture visualizzata tra due soffitti adiacenti che si trovano altezza differente;

Lower è la texture visualizzata tra due pavimenti adiacenti che si trovano altezza differente;

Il numero di parametri di ogni blocco è a questo punto salito a sedici.
Per chiarire meglio le idee, si osservi la seguente figura:



Come si può notare, il mondo risultante da questa nuova definizione, è anch'esso sostanzialmente bidimensionale. Il realismo è però sicuramente superiore. Ne è prova l'enorme successo riscosso da Doom.

L'applicazione ai BSP tree dei concetti descritti in questo paragrafo è semplice. Prima di tutto l'unità fondamentale non è costituita dai blocchi ma, ovviamente, dalle linee. Ogni linea rappresenta un muro ed è quindi dotata delle tre texture (upper, normal e lower). Le linee costituiscono i lati di poligoni detti settori. Ogni settore ha come parametri l'altezza e la texture di pavimento e soffitto.

Per un maggiore approfondimento consiglio la lettura delle specifiche dei file WAD di Doom, contenute nel file "DoomSpecs.guide".

La "tecnica del pittore"

Siamo ora a conoscenza delle due tecniche più usate per decidere cosa tracciare per visualizzare una scena. Utilizzando il ray-casting, ci ritroveremo con una lista di trattini verticali, ognuno relativo ad un muro, mentre utilizzando i BSP tree, ci ritroveremo con una lista di facce, scomponibili in trattini verticali.

[DA COMPLETARE]

30.7 Illuminiamo il nostro mondo

E' possibile dotare i blocchi (nel caso del ray-casting) o i settori (nel caso dei BSP) del parametro di illuminazione. Tale parametro viene utilizzato durante il texture mapping per accedere ad una tabella di illuminazione e variare la luminosità di ogni pixel delle texture del blocco.

Si supponga che le texture siano a 256 colori. Questo significa che ogni pixel di una qualunque texture può assumere valori tra 0 e 255. La palette di 256 colori deve essere composta da un certo numero di sfumature di un certo numero di colori di base (ad esempio 32 grigi, 32

marroni, 32 rossi, 16 blu, etc.), in modo da coprire un pò tutte le esigenze cromatiche di una qualunque immagine. Quindi ogni colore è presente più volte nella palette ma con intensità luminosa diversa.

E' quindi possibile costruire una tabella che associa ad ogni colore della palette un altro colore della palette stessa, ma con differente luminosità. Gli elementi di questa tabella assumono, ovviamente, valori tra 0 e 255.

Costruendo M tabelle di questo tipo, ognuna relativa ad una diversa luminosità (compresa tra 100% e 0%), si ottiene una matrice N x M, dove N è il numero di colori della palette (256). Con 32 livelli di illuminazione (e quindi 32 tabelle) si ottiene una matrice che occupa quindi 256*32=8192 bytes.

Supponendo che il colore 0 della palette sia il nero, la matrice può essere così presentata:

```
                              COLORI DELLA PALETTE

                     0   1   2   3  ............  253 254 255
                   +-----------------------------------------+
            100%   |  0   1   2   3  ............  253 254 255 |
             .     |  .   .   .   .  ............    .   .   . |
             .     |  .   .   .   .  ............    .   .   . |
    L        .     |  .   .   .   .  ............    .   .   . |
    U       75%    |  .   .   .   .  ............    .   .   . |
    M        .     |  .   .   .   .  ............    .   .   . |
    I        .     |  .   .   .   .  ............    .   .   . |
    N        .     |  .   .   .   .  ............    .   .   . |
    O       50%    |  .   .   .   .  ............    .   .   . |
    S        .     |  .   .   .   .  ............    .   .   . |
    I        .     |  .   .   .   .  ............    .   .   . |
    T        .     |  .   .   .   .  ............    .   .   . |
    A'      25%    |  .   .   .   .  ............    .   .   . |
             .     |  .   .   .   .  ............    .   .   . |
             .     |  .   .   .   .  ............    .   .   . |
             .     |  .   .   .   .  ............    .   .   . |
            0%     |  0   0   0   0  ............    0   0   0 |
                   +-----------------------------------------+
```

Come si può notare, la prima tabella (quella relativa alla luminosità 100%) fa corrispondere ad ogni colore della palette, il colore stesso, per cui in teoria si potrebbe evitare di accedere alla tabella. L'ultima tabella, relativa alla luminosità minima, fa invece corrispondere ad ogni colore della palette, il colore zero, cioè il nero. Le tabelle intermedie devono essere calcolate tramite una routine apposita. Allegato a questo articolo c'è un sorgente C adatto allo scopo. Il suo nome è: `MakeLTable.c`

Per gestire l'illuminazione, il ciclo di texture mapping va modificato in questo modo:

```
1         moveq   #0,d5
2  loop   move.b  (a0,d0.w),d5    ;Legge il pixel dalla texture
3         move.b  (a2,d5.l),(a1)  ;Legge dalla light table e scrive
4         add.w   d3,d1           ;Somma la parte frazionaria
5         addx.w  d2,d0           ;Somma la parte intera (+ riporto)
6         adda.l  d4,a1           ;Sposta il ptr. allo schermo
7         dbra    d7,loop
```

dove a2 è il puntatore alla tabella di illuminazione che deve ovviamente essere calcolato al di fuori del ciclo, tenendo conto della luminosità del mondo in quel punto e della distanza dall'osservatore. In un ambiente al massimo della luminosità, a2 punta sempre alla prima tabella, mentre in un ambiente totalmente buio punta all'ultima.

30.8 Texture mapping e Amiga

Penso sia noto a tutti che la Id, la casa di software che ha realizzato Doom, ha sempre creduto che fosse impossibile portare Doom su Amiga. Secondo loro, il 4000/40 sarebbe sufficientemente veloce per Doom se non fosse per la mancanza di una modalità grafica in chunky pixel e per l'elevato prezzo di tale macchina. Di realizzare Doom per un 1200, neanche a parlarne. Hanno ragione ? In parte si. Doom è realizzato principalmente in C, e solo una minima parte del codice è stata scritta in assembly, per cui solo i processori più veloci possono farlo girare ad una velocità decente. Riscriverlo interamente in assembly per l'Amiga non sarebbe affatto conveniente, per cui la Id ha preferito evitare di realizzare una conversione. Dal punto di vista della convenienza commerciale, quindi, la Id ha sicuramente ragione, ma dal punto di vista puramente tecnico ? Come afferma la Id, i problemi sono sostanzialmente due: la scarsa diffusione di processori veloci (e di conseguenza, di costo elevato) e la mancanza di un modo grafico in chunky pixel.

Contrariamente a quello che si crede, la mancanza del chunky pixel non è il problema principale del texture mapping su Amiga. Sono in molti a credere che i modi grafici planari siano ben otto volte più lenti dei corrispettivi modi in chunky pixel, ma questo non è del tutto vero. E' vero che se si vuole scrivere un solo pixel per volta in uno schermo planare, bisogna accedere otto volte alla memoria per scrivere ognuno degli otto bit che compongono il pixel, ma è pur vero che è in certe applicazioni non è affatto necessario scrivere un solo pixel per volta. Esistono infatti tecniche che permettono di ottimizzare l'accesso alla memoria video e che permettono di avvicinarsi in maniera soddisfacente alle prestazioni delle modalità grafiche in chunky pixel. E' questo il caso della conversione "chunky to planar".

Conversione Chunky to Planar

Questa tecnica prevede l'uso di un finto buffer in chunky pixel, che il programma tratta come se fosse un vero schermo in chunky pixel. Terminata l'elaborazione di un frame, basta eseguire una routine che si occupa di convertire, il più velocemente possibile, il finto schermo chunky pixel nello schermo planare visualizzato dal chip-set di Amiga. Esistono diverse tecniche (e varianti delle stesse) per effettuare la conversione, ed è possibile trovare nel pubblico dominio un certo numero di routine già pronte e complete di sorgenti. In generale si può dire che esistono due grandi famiglie di routine di conversione: quelle che utilizzano il blitter e quelle che non ne fanno uso. Le prime sono ottime per le macchine non troppo veloci, come il 1200 base o il 1200 con fast. Le seconde sono, invece, preferibili su macchine veloci, quelle dotate di 68030 50Mhz o di 68040. Cerchiamo di capire perchè.

E' risaputo che la memoria chip è molto più lenta della fast, sia perchè rallentata dagli accessi DMA, sia perchè funzionante con un clock di soli 7 Mhz. Il 68040 25Mhz, che potrebbe accedere alla RAM in soli 2 cicli (80ns), ha bisogno di qualcosa come 16 cicli (640ns) per accedere alla chip (vengono cioè inseriti 14 cicli di attesa): il collo di bottiglia è proprio li, nell'accesso alla chip RAM. Grazie alla pipeline , dopo una operazione di accesso in scrittura alla memoria, i processori dal 68020 in su possono passare subito ad eseguire altre istruzioni che non accedano alla memoria, senza attendere che la scrittura sia terminata, questo sempre che il codice da eseguire sia nella cache. Le istruzioni eseguite "all'ombra" dell'istruzione di accesso in scrittura alla memoria vengono comunemente chiamate "free instructions" o istruzioni gratuite. Data la maggiore velocità del 68040 rispetto al 68020 e tenuto conto del collo di bottiglia rappresentato dalla chip RAM, si nota facilmente come il 68040 sia in grado di eseguire un buon numero di istruzioni gratuite. Volendo fare un semplice esempio, l'insieme di istruzioni:

```
1 move.l d0,(a0)+
2 move.l d1,(a0)+
3
```

```
è veloce quanto il seguente:

 move.l d0,(a0)+
 add.l  d2,d0            <—— istruzione gratuita
 move.l d1,(a0)+
```

Il numero di istruzioni gratuite varia in dipendenza dal numero di cicli di attesa imposti al processore, e dalla velocità del processore stesso. Maggiori sono questi due parametri, maggiori sono le limitazioni imposte dal collo di bottiglia rappresentato dalla chip RAM. Di conseguenza il numero di istruzioni gratuite aumenta.

Tutto questo significa che insieme ad una semplice copia di dati dalla fast alla chip RAM, è possibile eseguire, su un processore veloce, anche un'elaborazione dei dati stessi, il tutto senza tempi aggiuntivi.

Purtroppo, per quanto riguarda la conversione chunky to planar, non è possibile scrivere tutto il codice in modo tale che tutte le istruzioni che non accedono alla memoria siano gratuite, ma è possibile andare vicino a questo risultato.

Come se non bastasse, sembra che il blitter sia sensibilmente più lento sulle macchine dotate di processori più veloci.

In conclusione, riporto un confronto dei tempi di esecuzione, su 1200 e 4000, della routine di chunky to planar che utilizzo per il 1200, e che sfrutta sia il blitter, sia il processore:

```
- A1200+fast:
        68020   :  41 msec
        Blitter :  66 msec
        ------------------
        Totale  : 107 msec

- A4000:
        68040   :  24 msec
        Blitter :  80 msec
        ------------------
        Totale  : 104 msec
```

Come si può notare, il Blitter del 4000 è sensibilmente più lento di quello del 1200, per cui le prestazioni globali di questa routine di chunky to planar sono quasi identiche tra 1200 e 4000. In verità sul 4000 le cose vanno comunque meglio che sul 1200, dato che la parte di conversione affidata al processore è di 17 msec più veloce.

Copper Chunky

Esiste un trucco per ottenere una specie di modalità grafica in chunky pixel su Amiga: si tratta del "copper chunky". Vediamo un pò di cosa si tratta. Sappiamo bene che il copper è in grado di modificare il contenuto dei registri colore e, quindi, dei pixel sullo schermo. Proviamo allora a scrivere una copper list che cambi il colore del fondo in questo modo:

```
 $0180, $0f00
 $0180, $0000
 $0180, $0f00
 $0180, $0000
 $0180, $0f00
 $0180, $0000
 $0180, $0f00
 $0180, $0000
 $0180, $0f00
 $0180, $0000
```

Come si può notare, questo pezzo di copper list alterna il rosso e il nero come colore di fondo. Si potrebbe allora accedere alla seconda word di ogni istruzione copper come se fosse un pixel di

uno schermo chunky. La velocità del copper impone però un limite costituito dal numero di pixel di diverso colore visualizzabili e dalla dimensione degli stessi. Infatti, la copper list dell'esempio, cambia il colore del fondo ogni 8 pixel in bassa risoluzione, per cui potremmo ottenere uno schermo chunky di soli 40 pixel di dimensioni 8x1: praticamente inservibile, sia per il ridotto numero di pixel, sia per le dimensioni degli stessi. Possiamo allora provare a modificare anche gli altri registri colore, per è necessario uno schermo che contenga, su ogni riga, qualcosa del genere:

```
Colore0,Colore1,Colore2,Colore3,......
```

La copper list corrispondente ad ogni riga dello schermo potrà quindi essere:

```
$0180, $0rgb
$0182, $0rgb
$0184, $0rgb
$0186, $0rgb
.....  .....
```

Purtroppo il copper non riesce ad eseguire più di una cinquantina di istruzioni per riga, per cui il numero di pixel del nostro schermo chunky risulterà essere comunque troppo basso. Questo però significa anche che in due righe, il copper può eseguire un centinaio di istruzioni e, quindi, modificare un numero soddisfacente di registri colore (poco meno di 100). Possiamo quindi aprire uno schermo a 7 bitplane, in cui ogni riga sia, ad esempio, del tipo:

```
Colore0,Colore0,Colore1,Colore1,Colore2,Colore2,...,Colore95,Colore95
```

e scrivere una copper list che, ogni due righe, modifichi il contenuto dei 96 registri colore (ovviamente utilizzando il registro BPLCON3=*$dff106* per modificare il banco dei registri colore). Avremo così realizzato un schermo copper chunky con pixel da 2x2 che però, purtroppo, non funziona ancora nel migliore dei modi. Infatti ogni pixel da 2x2 non appare di un unico colore, proprio perchè il copper non riesce a cambiare i registri colore in maniera sufficientemente veloce. Servirebbe una specie di double-buffering. Per fortuna ci viene in aiuto una caratteristica del chipset AGA costituita dalla possibilità di cambiare l'insieme di colori utilizzati per la visualizzazione. Ogni volta che lo hardware video deve visualizzare un pixel, deve leggerne il colore RGB da uno dei 256 registri colore. Infatti il valore scritto nei bitplane non è altro che un indice nella tabella dei registri colore. Prima di effettuare l'accesso ai registri colore, viene effettuato un OR esclusivo tra il valore letto dai bitplane e il contenuto degli 8 bit alti del registro `BPLCON4=$dff10c`. Gli 8 bit alti di `BPLCON4` vengono chiamati `BPLAMx` (dove x = 1-8). E' facile quindi capire che, ponendo `BPLAM=$80`, i colori visualizzati saranno quelli da 128 a 255, piuttosto che quelli da 0 a 127.

La copper list, quindi, sarà scritta in maniera da modificare i colori da 0 a 95, mentre sono visualizzati quelli da 128 in poi, e da modificare i colori da 129 a 224, mentre sono visualizzati quelli da 0 in poi:

```
$010c,$8000          ;visualizza i colori da 128 a 255
$0106,$0020          ;seleziona il primo banco di 32 colori
$0180,$0rgb          ;modifica il colore del registro 0
$0182,$0rgb          ;modifica il colore del registro 1
$0184,$0rgb          ;modifica il colore del registro 2
$0186,$0rgb          ;modifica il colore del registro 3
...........           ..............................
$01be,$0rgb          ;modifica il colore del registro 31
$0106,$2020          ;seleziona il secondo banco di 32 colori
$0180,$0rgb          ;modifica il colore del registro 32
$0182,$0rgb          ;modifica il colore del registro 33
$0184,$0rgb          ;modifica il colore del registro 34
$0186,$0rgb          ;modifica il colore del registro 35
...........           ..............................
```

```
15  $01be,$0rgb          ;modifica il colore del registro 63
16  $0106,$6020          ;seleziona il terzo banco di 32 colori
17  $0180,$0rgb          ;modifica il colore del registro 64
18  $0182,$0rgb          ;modifica il colore del registro 65
19  $0184,$0rgb          ;modifica il colore del registro 66
20  $0186,$0rgb          ;modifica il colore del registro 67
21  ...........           ..............................
22  $01be,$0rgb          ;modifica il colore del registro 95
23
24  $xx01,$fffe          ;attende la prossima riga da 2 pixel
25  $010c,$0000          ;visualizza i colori da 0 a 127
26  $0106,$8020          ;seleziona il quinto banco di 32 colori
27  $0180,$0rgb          ;modifica il colore del registro 128
28  $0182,$0rgb          ;modifica il colore del registro 129
29  $0184,$0rgb          ;modifica il colore del registro 130
30  $0186,$0rgb          ;modifica il colore del registro 131
31  ...........           ..............................
32  $01be,$0rgb          ;modifica il colore del registro 159
33  $0106,$a020          ;seleziona il sesto banco di 32 colori
34  $0180,$0rgb          ;modifica il colore del registro 160
35  $0182,$0rgb          ;modifica il colore del registro 161
36  $0184,$0rgb          ;modifica il colore del registro 162
37  $0186,$0rgb          ;modifica il colore del registro 163
38  ...........           ..............................
39  $01be,$0rgb          ;modifica il colore del registro 191
40  $0106,$e020          ;seleziona il settimo banco di 32 colori
41  $0180,$0rgb          ;modifica il colore del registro 192
42  $0182,$0rgb          ;modifica il colore del registro 193
43  $0184,$0rgb          ;modifica il colore del registro 194
44  $0186,$0rgb          ;modifica il colore del registro 195
45  ...........           ..............................
46  $01be,$0rgb          ;modifica il colore del registro 223
```

Un buon esempio della tecnica del copper chunky è contenuta nel file `chunky.lha` allegato a questo articolo.

CAPITOLO 31

REAL-TIME COMPUTER GRAPHICS 3D

Autore: Cristiano Tagliamonte Aceman/RAMJAM

31.1 Prefazione

Questo breve testo vuol essere di aiuto a chi vuol intraprendere l'arduo ed affascinante cammino verso la programmazione di un motore grafico 3D, scoprendone inizialmente i concetti di base per poi affrontare i più complessi e spettacolari effetti realizzabili. Col termine "motore" si indica l'insieme di routine atte alla gestione e alla manipolazione di specifici dati che una volta elaborati nella maniera dovuta daranno come risultato la visualizzazione dell'ambiente 3D in tempo reale.

Il seguente testo non vuol essere un corso di programmazione orientato alle applicazioni 3D, bensì un'opera nella quale vengano forniti più semplicemente i concetti sui quali si fondano molti motori 3D.

Il tutorial è stato suddiviso in parti, in cui ogni parte è composta da capitoli tra loro indipendenti, ovvero (esclusi alcuni casi in cui certi argomenti sono legati tra loro intrinsecamente) la comprensione di un argomento citato in un determinato capitolo non dipende dagli altri capitoli. In questo modo si rende più efficace ed immediata la consultazione del presente testo. Naturalmente un capitolo potrebbe richiedere la conoscenza di particolari argomenti, pertanto all'inizio di ognuno di essi vengono elencati eventuali titoli dei capitoli in cui vengono fornite le informazioni necessarie affinché sia possibile comprendere appieno gli argomenti trattati. Inoltre, nella lista degli argomenti necessari alla comprensione, potranno essere presenti eventuali sigle "n.p." atte ad indicare che tali argomenti, a causa della loro natura, non sono presenti nel seguente testo.

Nel caso il lettore non abbia le dovute conoscenze del linguaggio C si raccomanda di porgere priorità alla consultazione dell'Appendice A, nella quale vengono forniti chiarimenti riguardo la sintassi dello pseudo-codice utilizzato, ripresa appunto dal C.

Per concludere si raccomanda la conoscenza di base della trigonometria, dell'algebra lineare e di un linguaggio di programmazione (meglio se non troppo evoluto).

31.2 Parte 0: Cenni di grafica 2D relativa agli 80x86 e VGA

Introduzione

In questa sezione non approfondiremo l'utilizzo delle VGA e SVGA, bensì tratteremo le più elementari funzioni atte alla manipolazione di strutture bidimensionali, indispensabili per poter realizzare progetti in computer grafica 3D.

In altre parole verranno date le nozioni minime per poter comprendere ed applicare i concetti sulla computer grafica 3D, senza esporre in maniera peculiare le caratteristiche delle VGA/SVGA, il cui studio richiederebbe una documentazione dedicata, che non è l'obiettivo di questo tutorial. A tal proposito verrà analizzato il solo modo video 320*200 a 256 colori, il più immediato da gestire.

Tutti gli argomenti presentati in questa sezione richiedono espressamente la conoscenza dell'Assembly relativo agli 80x86 presumendo di lavorare in modalità protetta 32 bit con un modello di memoria di tipo flat.

Apertura di uno schermo 320*200 a 256 colori

Analizziamo questa manciata di istruzioni assembly:

```
mov     eax,13h         ; imposta EAX al cui valore corrisponde
                        ; la modalità  video da aprire. Nel nostro
                        ; caso 13h ci permette l'apertura di uno
                        ; schermo 320*200 a 256 colori.
int     10h             ; interrupt per la gestione della grafica.
```

Come possiamo vedere è molto semplice utilizzare una risoluzione 320*200 a 256 colori: basta eseguire giusto queste due istruzioni assembly e tale modo video viene aperto!

Nel caso volessimo ripristinare la vecchia modalità testo (standard del dos) non dovremo far altro che seguire queste istruzioni:

```
mov     eax,03h         ; imposta EAX al cui valore corrisponde
                        ; la modalità  video da aprire. Nel nostro
                        ; caso 03h ci permette l'apertura di uno
                        ; schermo in modalità  testo 80*25.
int     10h             ; interrupt per la gestione della grafica.
```

Questa coppia di istruzioni dovranno essere eseguite appena prima dell'uscita all'MS-DOS del nostro programma.

Organizzazione della memoria video

Uno schermo 320*200 a 256 colori ha come indirizzo fisico di partenza *000A0000h*. A partire da tale indirizzo ogni byte rappresenterà un pixel visualizzato su schermo e di conseguenza, dato che un byte può coprire un range di valori da 0 a 255, potremo utilizzare al massimo 256 colori.

Il byte corrispondente l'indirizzo *000A0000h* è il pixel estremo superiore a sinistra, mentre ai relativi pixel adiacenti corrisponderanno i byte immediatamente successivi a all'indirizzo *000A0000h*. Il 321esimo byte (a partire da *000A0000h*, quindi *000A0140h*) risulta il pixel estremo sinistro della seconda riga. Vediamo un semplice schema per chiarire le idee:

```
+---- 000A0001h             . = ipotetico pixel sul monitor
|+--- 000A0002h
||+-- 000A0003h
vvv
```

```
000A0000h -> ........................... <- 000A013Fh
000A0140h -> ........................... <- 000A027Fh
000A0280h -> ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
             ...........................
000AF8C0h -> ........................... <- 000AF987h
```

E' possibile notare che l'area di memoria interessata alla visualizzazione dei pixel è compresa tra *000A0000* e *000AF987*, che corrisponde a 64000 byte (difatti 320*200=64000).

Tracciamento di un pixel

Per visualizzare un pixel conoscendone le coordinate X e Y dovremo calcolare la giusta locazione di memoria su cui è posto in memoria. Per far ciò occorrerà utilizzare una legge matematica che a partire dalle coordinate X e Y ricavi l'indirizzo effettivo del pixel, ovvero:

```
x        <- coordinata x del pixel (range: da 0 a 319)
y        <- coordinata y del pixel (range: da 0 a 199)
destaddr <- indirizzo fisico effettivo del pixel
destaddr = 000A0000 + 320*y + x
```

La formula appena vista permette sì un corretto calcolo dell'indirizzo fisico del pixel, ma non la si può propriamente definire ottimizzata in quanto è presente una moltiplicazione; cerchiamo di sostituirla con qualche operazione meno pesante:

```
320*y = (256 + 64)*y = 256*y + 64*y = y<<8 + y<<6
destaddr = 000A0000 + y<<8 + y<<6 + x
```

Abbiamo eliminato la moltiplicazione utilizzando due snelli shift a sinistra (256 e 64 sono potenze di 2) che rendono il nostro codice sicuramente più efficiente.

Settare la palette dei colori

Abbiamo già detto che ogni byte rappresenta il colore di un pixel, tale byte si definisce "chunky pixel". Questo byte non è altro che un indice di una tabella dove sono indicati i 256 colori da utilizzare. Infatti l'insieme dei colori esistenti nella nostra risoluzione (320*200) sono 262144, di cui possiamo sceglierne 256 da poter visualizzare. Analizziamo dove e come definire tale tabella.

Ogni colore idealmente realizzabile può essere scomposto da tre componenti: il rosso, il verde e il blu. Mescolando questi tre colori secondo intensità differenti è possibile ottenere tutti i colori rappresentabili dalla nostra VGA.

Nel nostro caso, per dimensionare un qualsiasi colore, definiremo le relative componenti R (red), G (green) e B (blue).

L'insieme delle componenti RGB rappresenta il colore che vogliamo ottenere. Una componente R, G o B non è altro che un byte in cui vengono presi in considerazione i 6 bit meno significativi; in altre parole per ogni componente abbiamo a disposizione un range di valori compreso tra 0 e 63, ovvero 64 combinazioni (infatti 64*64*64=262144 ovvero l'intera gamma di colori a disposizione). Un valore basso indica una bassa intensità (tonalità scura), al contrario più il valore sarà alto e maggiore sarà l'intensità di quella componente (tonalità chiara).

Per settare la palette (ovvero la tavolozza di colori che si vuol utilizzare), ci serviranno solo le due porte hardware di indirizzo *03C8h* e *03C9h*. Ecco la procedura da seguire per settare un colore:

- scrivere nella porta *03C8h* un byte, il quale indicherà quale colore andremo a settare (0 è il primo colore, 255 è l'ultimo);
- scrivere nella porta *03C9h* il byte contenente la componente R (rosso);
- scrivere nella porta *03C9h* il byte contenente la componente G (verde);
- scrivere nella porta *03C9h* il byte contenente la componente B (blu).

Da notare che una volta settato un colore si potrà continuare a scrivere nella porta *03C9h* i successivi colori senza dover memorizzare (di volta in volta) il numero del colore successivo nella porta *03C8h*.

Tutto ciò tradotto in assembly porta ad un codice che può essere scritto nella seguente forma:

```
        lea     esi,[palette]   ; puntatore alla tabella dei colori
        mov     edx,03C8h       ; porta 03C8h
        xor     eax,eax         ; puliamo eax
        out     dx,al           ; iniziamo a settare il primo colore
        inc     edx             ; porta 03C9h
        mov     ecx,256         ; numero di iterazioni del loop
@@loop:
        mov     al,[esi]        ; leggiamo la componente Red...
        out     dx,al           ; ... e la memorizziamo
        mov     al,[esi+1]      ; leggiamo la componente Green...
        out     dx,al           ; ... e la memorizziamo
        mov     al,[esi+2]      ; leggiamo la componente Blue...
        out     dx,al           ; ... e la memorizziamo
        add     esi,3           ; prossimo colore da settare
        dec     ecx
        jnz     @@loop
        ...
        ...                     ; altro codice
        ...

palette:                        ; la tabella dei colori
        db      00, 00, 00      ; colore 00h : R, G, B
        db      11, 23, 45      ; colore 01h : R, G, B
        ...
        ...                     ; altri colori della tavolozza
        ...
        db      63, 63, 63      ; colore FFh : R, G, B
```

Il sorgente appena esaminato non fa altro che leggere i byte nella tabella dei colori a partire dall'indirizzo "palette" e li scrive nella porta *03C9h* non prima di aver precisato nella porta *03C8h* che intendiamo iniziare a settare la palette a partire dal colore *00h*, fino ad arrivare all'ultimo, ovvero il colore *FFh*.

Sincronismo del processore col refresh video

Utilizzando una risoluzione come la 320*200 a 256 colori, lo schermo viene aggiornato 70 volte al secondo (70 Hz). In altre parole quel che si visualizza sul monitor viene tracciato 70 volte al secondo, ciò permette l'illusione di realizzare animazioni fluide. Il refresh non è che il procedimento fisico per cui lo schermo viene aggiornato (illuminando in maniera opportuna ogni pixel).

Mettiamo caso di voler realizzare l'animazione di un oggetto 3D renderizzato in tempo reale. Per rendere il movimento di tale oggetto più fluido possibile dovremo calcolare un intero frame in un 70esimo di secondo. Consideriamo che le routine che abbiamo realizzato siano abbastanza ottimizzate da permettere che l'elaboratore calcoli oltre 70 fotogrammi per secondo. In questo caso dovremo scrivere una routine che, una volta renderizzato un frame, permetta al processore di attendere il termine del refresh video prima di calcolare il prossimo fotogramma. Questo è ciò che si intende con "sincronizzazione del processore col refresh video".

Ora vediamo come realizzare il tutto in Assembly evitando di analizzare dettagliatamente il codice:

```
        mov     edx,3DAh
@@wait_refresh1:
        in      al,dx
        test    al,8
        jz      @@wait_refresh1
@@wait_refresh2:
        in      al,dx
        test    al,8
        jnz     @@wait_refresh2
```

In questo spezzone di codice il processore non fa altro che attendere (grazie al primo loop) che il pennello elettronico (che aggiorna lo schermo) giunga ad una ben determinata linea video, quindi (nel secondo loop) si aspetta che tale linea venga completamente tracciata.

Questa tecnica viene applicata per evitare di calcolare un nuovo fotogramma ancor prima di averlo visualizzato.

31.3 Parte 1: Geometry Engine

Introduzione

In questa sezione studieremo la geometria che si cela dietro la computer grafica 3D, in quanto ogni elemento elaborato da un motore 3D consiste in un insieme di valori numerici che attribuiscono una o più proprietà geometriche, quindi il movimento compiuto da un solido proiettato su monitor è frutto di opportune trasformazioni geometriche.

Trasformazioni prospettiche

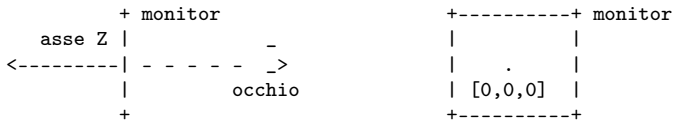
Concetto di prospettiva

Riguardo le nostre applicazioni (relative alla realizzazione di un motore 3d), alla base della prospettiva risiede il principio con cui una qualsiasi entità (punto, linea, poligono o oggetto che sia) da uno spazio tridimensionale possa essere rappresentata in uno spazio a due dimensioni. Difatti uno dei primissimi problemi che intercorre nella scrittura di un engine 3d è come visualizzare tali entità su uno spazio 2d, ossia il monitor, quando all'origine ogni cosa viene rappresentata matematicamente in uno spazio 3d. Per far fronte a questa esigenza viene in aiuto la prospettiva che ci permette si applicare la "conversione" da uno spazio 3d ad uno spazio 2d.

Innanzitutto va sottolineato che lo studio di trasformazioni prospettiche sarà limitato ai soli vettori. Infatti ogni entità è derivabile da un insieme ordinato di uno o più vettori (un punto è rappresentabile con un vettore; una linea si può rappresentare matematicamente come quella coppia di vettori coincidenti con gli estremi della linea stessa; un poligono è una sequenza ordinata di vertici, i quali possono essere indicati con vettori; infine un oggetto è un insieme ben definito di poligoni).

Dati un punto di vista e un piano di proiezione perpendicolare alla direzione di osservazione e posto di fronte al suddetto punto vista, il concetto di prospettiva è di tracciare un raggio passante sul punto di vista e sul vettore 3d di cui si vuol conoscere la proiezione sul piano (questo avviene esclusivamente a livello teorico, a livello pratico non si traccia nessun raggio!); quindi il vettore coincidente con il punto di intersezione tra il raggio ed il piano, rappresenterà la proiezione di quel vettore.

Dato un ipotetico punto di vista e un piano di proiezione (rispetto ai quali dovremo ricavare le coordinate 2d), consideriamo che la direzione di osservazione sia perpendicolare al nostro piano e che tale piano coincida con il piano XY del nostro spazio 3d (ovvero il piano di equazione z = 0). Inoltre la retta coincidente con la direzione di osservazione passa per l'origine dei tre assi del nostro sistema di riferimento XYZ. Questo implica che tale retta coincide con l'asse z, sul quale sarà quindi posto il punto di vista. Il monitor è inteso come quella parte (ovvero quel sottospazio) di piano visibile. Vediamo praticamente la nostra situazione:

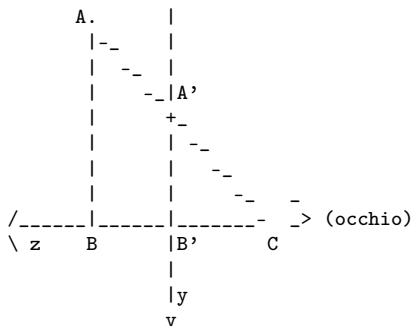


Questa situazione è molto importante, in quanto permette di semplificare concettualmente e algoritmicamente le trasformazioni prospettiche.

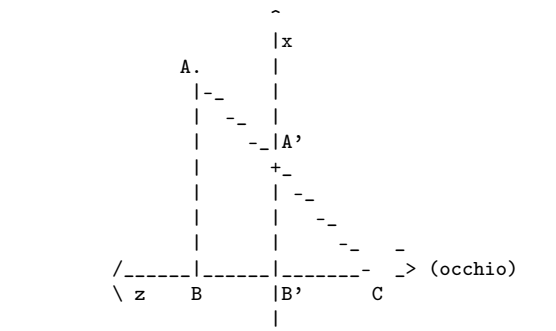
Implementazione della prospettiva

Abbiamo un vettore di cui conosciamo le coordinate $[x, y, z]$. Il nostro obiettivo consiste nel calcolare le coordinate $[xs, ys]$ proiettate sullo schermo, quindi dobbiamo applicare una trasformazione prospettica.

Consideriamo un punto, denominato A, nello spazio visibile dal video. Facciamo finta che il nostro monitor sia trasparente e che quel punto sia davvero presente nello spazio. Ora, la nostra situazione vista lateralmente è la seguente:



Mentre questo è il caso analogo visto dall'alto:




```
A  = vettore nello spazio [x, y, z] rappresentante il punto a cui
     si vuol applicare la trasformazione prospettica;
B  = vettore avente la stessa coordinata z del vettore A, ma con
     con coordinate x e y uguali a zero, è  del tipo [0, 0, z];
C  = vettore coincidente con il punto vista, per comodità  è
     posto sull'asse z (quindi ha le coordinate x,y = 0);
A' = vettore coincidente con la proiezione su video del vettore A,
     è  della forma [xs, ys, 0];
B' = vettore avente la stessa coordinata z del vettore A', ma con
     con coordinate x e y uguali a zero, è  esattamente [0, 0, 0],
     ciò significa che B' coincide con l'origine degli assi;
```

Si ricorda, come precedentemente definito, che il piano di proiezione è quello di equazione $z = 0$, ovvero il piano XY, su cui coincide, appunto, il monitor. Quindi sarà chiaro perché le coordinate $[xs, ys]$ sono esattamente le coordinate 2d rappresentanti il nostro vettore 3d su schermo.

Vediamo adesso come ricavare queste coordinate. Sia nella vista dall'alto che in quella laterale i triangoli ABC e $A'B'C$ sono simili in quanto hanno un angolo in comune ed entrambi hanno un angolo retto, quindi possiamo scrivere la seguente proporzione:

```
A'B' / AB = B'C / BC
```

che corrisponde a:

```
A'B' = AB * B'C / BC
```

Possiamo scrivere BC come $BB' + B'C$. Dato che B' coincide con l'origine, la distanza BB' equivale alla coordinata z del vettore B. Inoltre definiamo d come la lunghezza del lato $B'C$. La nostra formula si traduce nella seguente:

```
A'B' = d * AB / (z + d)
```

Rispetto alla vista laterale AB coincide con la coordinata y di A, mentre $A'B'$ coincide con ys; sostituendo questi valori otteniamo proprio la formula per calcolare la y proiettata su schermo:

```
ys = d * y / (z + d)
```

Nel caso della vista dall'alto il discorso è analogo, ma relativo alla coordinata x proiettata:

```
xs = d * x / (z + d)
```

Il valore d rappresenta la distanza tra il punto di vista e l'origine; e come tale può essere fissato arbitrariamente. Un buon valore con cui istanziare d è , ad esempio, 256.

Bisogna tener conto che nel video le coordinate hanno come origine il punto in alto a sinistra, mentre per la trasformazione 3d->2d sono state considerate al centro del monitor. Per ovviare questo problema basta sommare, al termine dei calcoli per la proiezione, una costante a xs e a ys grazie alle quali i relativi assi verranno traslati di un numero di pixel equivalente alla costante. Riassumendo, ponendo $d = 256$, le coordinate proiettate sullo schermo sono equivalenti a:

```
xc = larghezza schermo/2, per traslare l'ascissa a destra
yc = altezza schermo/2, per traslare l'ordinata in basso

xs = 256 * x / (z + 256) + xc
ys = 256 * y / (z + 256) + yc
```

Attenzione al fatto che la coordinata z del punto non può coincidere con il punto di vista dato che si verificherebbe una divisione per zero. Nemmeno si deve porre la profondità di un punto inferiore a quella del punto di vista: sarebbe assurdo poter visualizzare punti posti dietro l'osservatore!

Finalmente siamo in grado di svolgere trasformazioni prospettiche, che tuttavia risultano veramente semplici da applicare.

Traslazione

Applicazione delle traslazioni

Dato un vettore V che rappresenta un punto nello spazio, traslare V rispetto ad un vettore T significa trovare un ulteriore vettore, chiamiamolo W, che rappresenta il vettore V spostato, in direzione coincidente con il verso di T, di uno spazio equivalente alla lunghezza di T.

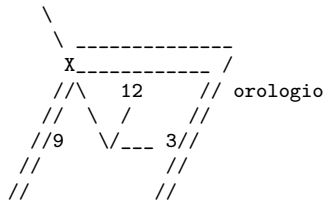
Tutto ciò si traduce in una banalissima somma tra vettori, più precisamente tra il vettore V e T, quindi:

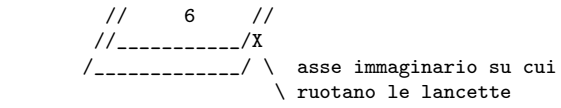
```
V[xv, yv, zv] = vettore da traslare
T[tx, ty, tz] = vettore coincidente la traslazione da applicare
W[xw, yw, zw] = vettore coincidente V traslato rispetto a T

W = V + T

xw = xv + tx
yw = yv + ty
zw = zv + tz
```

Rotazione

Concetto di rotazione

Consideriamo un orologio analogico, le sue lancette ruotano intorno l'asse passante per il centro dell'orologio e perpendicolare l'orologio stesso:





Tecnicamente parliamo di rotazione di un punto su di un asse, quando il punto si muove sul piano appartenente al punto stesso e perpendicolare all'asse in modo da non alterare la distanza punto-asse (che rimane costante). In questo modo il punto compie un movimento circolare intorno l'asse, ossia ci ruota intorno, e la distanza punto-asse funge da raggio per la circonferenza descritta dalla rotazione del punto in questione.

La rotazione intorno l'asse y si può immaginare come la traiettoria percorsa da un punto che giri intorno l'asse y senza modificare la propria ordinata, che rimane appunto inalterata.

31.4 Appendici

Appendice D: coordinate polari

Introduzione

Le coordinate polari vengono utilizzate in questo tutorial al fine di comprendere il principio che sta alla base dell'applicazione delle rotazioni.

Appendice G: cenni di algebra lineare

Introduzione

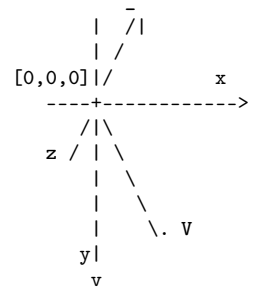
La seguente appendice è orientata a tutti coloro che non possiedono le nozioni di algebra lineare necessarie alla comprensione del testo ed in particolare della parte 1, relativa al geometry engine.

I vettori

Argomenti necessari alla comprensione: cenni di geometria piana (n.p.)

Definizione di un vettore

Un vettore non è altro che un oggetto matematico che rappresenta una quantità di valore con una direzione e un verso, o in termini spicci una linea. Nel contesto verranno sempre specificati vettori che vanno da un punto di coordinate $[0, 0, 0]$ (l'origine degli assi) verso un altro punto $[x, y, z]$, così facendo si può affermare che questo vettore ha come quantità $[x, y, z]$.



Un vettore viene indicato con una lettera, ad esempio nella precedente figura è rappresentato il vettore $V[x, y, z]$. Il sistema di riferimento usato è dato da tre variabili, immaginabile come un semplice piano cartesiano perpendicolare all'asse z. Per semplicità consideriamo la crescita della y verso il basso (e non verso l'alto) in modo da ridurre i calcoli da svolgere per la visualizzazione di un punto sullo schermo. L'asse z cresce quando esso si allontana dall'osservatore, mentre diminuisce nel caso si avvicini all'osservatore.

Utilizzeremo i vettori per definire ogni punto nello spazio (e in particolare i vertici di ogni oggetto 3d), ciò non significa in termini pratici che un vettore costituisca una linea visualizzata su schermo, bensì indica il segmento immaginario compreso tra l'origine degli assi ed un punto nello spazio.

Operazioni tra vettori

Innanzitutto va specificato che faremo uso fondamentalmente di vettori 2d e 3d in quanto il nostro motore gestirà punti nello spazio piano (per il tracciamento di poligoni su monitor) e tridimensionale (per eventuali trasformazioni geometriche), quindi vettori del tipo:

```
v2 = [ x , y ]        <---  generico vettore 2d
v3 = [ x , y , z ]    <---  generico vettore 3d
```

Tutte le operazioni tra vettori sono possibili solo tra vettori che hanno la stessa dimensione, quindi una qualsiasi operazione tra un vettore 2d ed uno 3d sono impossibili. Di seguito vediamo algebricamente quali sono queste operazioni:

ADDIZIONE il risultato della somma di due vettori è ancora un vettore della stessa dimensione dei due operandi:

```
v1 = [ x1 , y1 , z1 ]
v2 = [ x2 , y2 , z2 ]

v3 = v1 + v2 = [ x1+x2 , y1+y2 , z1+z2 ]
```

dove nel nostro caso $x1$, $x2$, $y1$, $y2$, $z1$ e $z2$ sono ovviamente numeri reali (che possono essere rappresentati come numeri in virgola fissa oppure preferibilmente come numeri in virgola mobile).

Da notare che il vettore $[0, 0, 0]$ coincide con l'operando nullo di questa operazione, ovvero addizionando tale vettore ad un qualsiasi vettore V otteremo come risultato sempre il vettore V. Inoltre tale operazione gode della proprietà commutativa.

Fisicamente è possibile immaginare la somma di due vettori 2d nella seguente maniera:

```
v1 = [ x1 , y1 ]
v2 = [ x2 , y2 ]
```

Ora immaginiamo di trovarci sul vettore $v1$. Quindi il risultato della somma $v1 + v2$ si può intendere come il risultato dello spostamento (dalla posizione indicata da $v1$) di $x2$ posizioni orizzontali e $y2$ posizioni verticali, questo implica la nostra posizione finale coinciderà col vettore $[x1 + x2, y1 + y2]$, ovvero il risultato dell'addizione.

NEGAZIONE il risultato della negazione di un vettore $v1$ è ancora un vettore della stessa dimensione di $v1$ e che è equidistante dall'origine rispetto a $v1$, ma è di verso opposto (in altre parole è simmetrico rispetto alla retta passante per l'origine, la quale è perpendicolare al vettore $v1$):

```
v1 = [ x , y , z ]
v2 = - v1 = [ -x , -y , -z ]
```

Anche in questo caso l'elemento nullo di questa operazione coincide con il vettore dell'origine $[0, 0, 0]$.

PRODOTTO CON UN FATTORE il prodotto tra un vettore $v1$ ed un fattore (inteso come un numero reale o in virgola mobile) è ancora un vettore delle stesse dimensioni di $v1$:

```
v1 = [ x , y , z ]
k  = numero reale

v2 = k * v1 = [ k*x , k*y , k*z ]
```

Il fattore k viene chiamato "scalare" (da non confondere con il prodotto scalare!), ed il vettore risultante v2 conserverà lo stesso verso di $v1$ ma la sua dimensione verrà scalata rispetto al valore di k. Questa operazione è commutativa.

PRODOTTO SCALARE il risultato prodotto scalare tra due vettori è uno scalare (quindi un numero reale o in virgola mobile):

```
v1 = [ x1 , y1 , z1 ]
v2 = [ x2 , y2 , z2 ]

k  = v1 * v2 = x1*x2 + y1*y2 + z1*z2
```

Geometricamente il prodotto scalare può essere definito come il prodotto tra le distanze dei due vettori con l'origine ed il coseno dell'angolo compreso tra i due vettori. Anche questa operazione gode della proprietà commutativa.

PRODOTTO IN CROCE il risultato del prodotto in croce tra due vettori è uno scalare (quindi un numero reale o in virgola mobile):

```
v1 = [ x1 , y1 ]
v2 = [ x2 , y2 ]

k  = v1 x v2 = x1*y2 - x2*y1
```

Per semplicità citiamo solo la formula del prodotto in croce tra due vettori 2d dato che utilizzeremo questa operazione solo su tali vettori. Il prodotto in croce non gode della proprietà commutativa.

Vettore unitario

Un vettore unitario è un vettore cui lunghezza pari all'unità . La lunghezza di un vettore è data dalla distanza del punto di coordinate specificate dal vettore stesso con l'origine.

La lunghezza di un generico vettore si ricava dal teorema di Pitagora svolgendo la radice quadrata della sommatoria dei quadrati delle componenti del vettore, più semplicemente:

```
v2 = [ x2 , y2 ]
v3 = [ x3 , y3 , z3 ]

l2 = |v2| = sqrt(x2*x2 + y2*y2)
l3 = |v3| = sqrt(x3*x3 + y3*y3 + z3*z3)
```

Rendere un generico vettore v1 come vettore unitario significa trovare quel vettore che conserva lo stesso verso di v1 ma che ha lunghezza pari ad 1. Nella pratica si traduce nel dividere ogni componente del vettore per la relativa lunghezza; in relazione al precedente esempio possiamo affermare:

```
u2 = [ x2/l2 , y2/l2 ]
u3 = [ x3/l3 , y3/l3 , z3/l3 ]
```

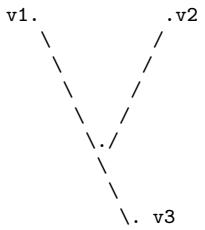
Che coincide col moltiplicare il vettore per uno scalare equivalente all'inverso della lunghezza del vettore:

```
v = [ x , y , z ]
k = 1 / |v| = 1 / sqrt(x*x + y*y + z*z)

u = k * v
```

Vettore normale

Un vettore 3d normale su di un piano (quindi su uno spazio bidimensionale) significa che la retta passante per tale vettore è perpendicolare a quel piano. Un vettore normale su una retta r significa che la retta passante per tale vettore è perpendicolare alla retta r. Un vettore normale ad altro vettore implica che il primo vettore sia normale alla retta passante per il secondo vettore.

Vediamo un esempio di vettore normale ad un altro vettore:



In questo caso $v1$ e $v3$ sono vettori normali rispetto a $v2$, mentre quest'ultimo è vettore normale sia rispetto a $v1$ che a $v3$. Infatti i vettori $v1$ e $v3$ giacciono sulla stessa retta.

Le matrici

Argomenti necessari alla comprensione: i vettori.

Definizione di matrice

Una matrice è una tabella di elementi (che nel nostro caso sono numeri razionali) del tipo:

```
          _                              _
         |                                |
         |  a(1,1)  a(1,2)  ...  a(1,m)  |
         |  a(2,1)  a(2,2)  ...  a(2,m)  |
M(n,m) = |  ...                           |
         |  ...                           |
         |  ...                           |
         |  a(n,1)  a(n,2)  ...  a(n,m)  |
         |_                              _|
```

dove $a(i,j)$ è il generico elemento della matrice mentre $n*m$ rappresenta la dimensione della matrice, in cui valgono le seguenti regole:

```
n, m > 0
0 < i <= n
0 < j <= m
```

n, m, i, j sono quindi numeri interi maggiori di zero. Il valore n è il numero di righe della matrice, mentre m è il numero di colonne. I valori i e j rappresentano rispettivamente gli indici di riga e di colonna di un elemento della matrice.

Nel nostro caso intendiamo ogni elemento $a(i,j)$ come un numero reale o un numero in virgola mobile.

Per semplicità si farà spesso riferimento ad una matrice denotando solo la lettera che la etichetta. Ovvero spesso si utilizzerà spesso la notazione M al posto di $M(n,m)$.

Casi particolari di matrici:

MATRICE QUADRATA M(n,n) il numero di righe e di colonne sono uguali. Esempio:

```
          _       _
         |         |
         | 1  2  3 |
M(n,n) = | 4  5  6 |
         | 7  8  9 |
         |_       _|
```

La DIAGONALE principale della matrice quadrata è definita come quell'insieme di elementi della matrice stessa tali che gli indici di riga e colonna siano uguali, quindi:

```
D = tutti gli a(i,i) per 0 < i <= n
```

ovvero:

```
D = { a(1,1) , a(2,2), ... , a(n,n) }
```

nel precedente esempio la diagonale principale è uguale a:

```
D = { 1 , 5 , 9 }
```

MATRICE TRASPOSTA $Mt(n,n)$ data una matrice quadrata $M(n,n)$, la sua trasposta $Mt(n,n)$ è quella matrice che contiene gli stessi elementi della M, ma con diverso ordine. Tale ordine è determinato dagli indici di riga e colonna, i quali vengono scambiati. Più formalmente il generico elemento $a(i,j)$ appartenente a M coincide con l'elemento $t(i,j)$ di Mt tale che:

```
t(i,j) = a(j,i)
0 < i <= n
0 < j <= n
```

Per chiarire le idee vediamo un esempio pratico relativo ad una matrice di dimensione 3*3:

```
    _       _                                _       _
   |         |                              |         |
   | 1  2  3 |                              | 1  4  7 |
M = | 4  5  6 |   la sua trasposta è    Mt = | 2  5  8 |
   | 7  8  9 |                              | 3  6  9 |
   |_       _|                              |_       _|
```

Da notare che gli elementi posti sulla diagonale principale rimangono nella posizione originaria.

MATRICE RIGA $M(1,m)$ una matrice riga è coincide, a livello logico, con la definizione di vettore, quindi tale matrice è intesa come una matrice con una sola riga, quindi nella forma:

```
M(1,m) = [ a(1,1)  a(1,2)  ...  a(1,m) ]
```

o più semplicemente:

```
M(m) = [ a(1)  a(2)  ...  a(m) ]
```

Le ultime due definizioni di matrice per righe sono logicamente equivalenti, la differenza sostanziale sta nel secondo caso, in cui non si fa specificatamente riferimento alla notazione matriciale, la quale implica che si faccia riferimento ad entrambi gli indici di riga e colonna (infatti abbiamo il solo indice di colonna).

Etichettando gli elementi di una matrice per riga senza il relativo indice si ottiene un oggetto matematico equivalente alla matrice per riga che viene denominato *ennupla*. Vediamone un esempio:

```
E = ( a , b , c , d )
```

MATRICE COLONNA $M(n,1)$ è un modo alternativo (rispetto alla matrice riga) di rappresentare un vettore secondo la notazione matriciale, ovvero:

```
          _       _
         |         |
         |  a(1,1) |
         |  a(2,1) |
M(n,1) = |  ...    |
         |  ...    |
         |  ...    |
         |  a(n,1) |
         |_       _|
```

o più semplicemente:

```
        _     _
       |       |
       |  a(1) |
       |  a(2) |
M(n) = |  ...  |
       |  ...  |
       |  ...  |
       |  a(n) |
       |_     _|
```

MATRICE NULLA $M(n,m)$ è quella matrice in cui tutti gli elementi sono uguali a 0, ovvero:

```
a(i,j) = 0
0 < i <= n
0 < j <= m
```

MATRICE IDENTITÀ $I(n)$ è un particolare caso di matrice quadrata in cui tutti gli elementi sono uguali a 0 fatta esclusione per quegli elementi presenti sulla diagonale principale che sono uguali a 1. O in termini più formali:

```
0 < i <= n
0 < j <= n
a(i,j) = 0   per i diverso da j
a(i,j) = 1   per i uguale a j
```

Faremo riferimento alla generica matrice identità $n * n$ con la notazione $I(n)$ (basta specificare un solo indice in quanto la matrice identità è sempre quadrata, quindi l'indice di colonna è sottointeso). Vediamo un esempio relativo alla matrice identità 4*4:

```
       _          _
      |            |
      | 1  0  0  0 |
I(4) = | 0  1  0  0 |
      | 0  0  1  0 |
      | 0  0  0  1 |
      |_          _|
```

Operazioni tra matrici

In relazione all'obiettivo di realizzare un motore 3d possiamo anticipare che principalmente ci occuperemo di applicare operazioni tra matrici 3*3, 4*3 e 4*4.

SOMMA la somma tra due matrici A e B è applicabile se e solo se A e B hanno lo stesso numero di righe e colonne. Il risultato è ancora una matrice, che per convenzione chiamiamo C, delle stesse dimensione degli operandi. Ogni elemento di C è la somma del corrispondente elemento di A e di B (che quindi si trovano nella stessa posizione dell'elemento di C):

```
A(n,m) = matrice cui generico elemento è  a(i,j)
B(n,m) = matrice cui generico elemento è  b(i,j)
C(n,m) = matrice cui generico elemento è  c(i,j)

c(i,j) = a(i,j) + b(i,j)
0 < i <= n
0 < j <= m
```

```
            _                                                 _
           |                                                   |
           | a(1,1)+b(1,1)  a(1,2)+b(1,2)  ...  a(1,m)+b(1,m) |
           | a(2,1)+b(2,1)  a(2,2)+b(2,2)  ...  a(2,m)+b(2,m) |
C = A + B = | ...                                               |
           | ...                                               |
           | ...                                               |
           | a(n,1)+b(n,1)  a(n,2)+b(n,2)  ...  a(n,m)+b(n,m) |
           |_                                                 _|
```

Questa operazione gode delle seguenti proprietà :

```
commutativa  =>  A + B = B + A
associativa  =>  (A + B) + C = A + (B + C)
```

NEGAZIONE la negazione di una matrice $A(n, m)$ è una matrice di dimensione ancora n*m in cui ogni elemento di A è di segno opposto:

```
A(n,m) = matrice cui elementi sono a(i,j)
B(n,m) = matrice cui elementi sono b(i,j)

B = - A

b(i,j) = - a(i,j)
0 < i <= n
0 < j <= m
```

PRODOTTO PER UNO SCALARE svolgere il prodotto di una matrice A per uno scalare k significa moltiplicare ogni elemento di A con k, il quale non è altro che un numero reale. La matrice così ottenuta è ovviamente delle stesse dimensioni della matrice originaria:

```
k      = numero reale
A(n,m) = matrice cui elementi sono a(i,j)
B(n,m) = matrice cui elementi sono b(i,j)

B = k * A

b(i,j) = k * a(i,j)
0 < i <= n
0 < j <= m
```

PRODOTTO TRA MATRICI il prodotto tra due matrici A e B è applicabile se il numero di colonne di A è uguale al numero di righe di B. Il risultato sarà una matrice C con un numero di righe equivalente a quello di A ed un numero di colonne uguale al corrispondente di B. Ogni elemento $c(i,j)$ di C coincide con il prodotto scalare dei due vettori intesi come la i-esima riga di A e la j-esima colonna di B:

```
A(n,t) = matrice cui generico elemento è  a(i,k)
B(t,m) = matrice cui generico elemento è  b(k,j)
C(n,m) = matrice cui generico elemento è  c(i,j)

C = A * B

c(i,j) = a(i,1)*b(1,j) + a(i,2)*b(2,j) + ... + a(i,t)*b(t,j)
0 < i <= n
0 < j <= m
```

che equivale a:

```
c(i,j) = a(i,k)*b(k,j)
0 < k <= t
0 < i <= n
0 < j <= m
```

Questa operazione gode della proprietà associativa, ovvero:

```
(A * B) * C = A * (B * C)
```

che, come vedremo in seguito, risulterà particolarmente utile per la progettazione del geometry engine. Inoltre il prodotto NON gode della proprietà commutativa. Da notare che l'elemento neutro di questa operazione è esattamente la matrice identità :

```
A(n,m) = I(n) * A(n,m) = A(n,m) * I(m)
```

Inoltre il prodotto per la matrice identità è l'unico caso in cui vale la proprietà commutativa.

31.5 Note finali

Spesso vagheggiano nella mia mente pensieri del tipo "Ne vale davvero la pena consumare tanto tempo per realizzare un motore 3d?", oppure "Ora c'è da fare altro di più prioritario che starsene qui a scrivere questo engine"..., dubbi che affliggono molte di quelle persone che si propongono di programmare un motore 3d, in particolare chi è alle prime armi oppure chi ne ha avuto a che fare per molto (forse troppo) tempo.

Tristo è quel discepolo che non avanza il suo maestro. Leonardo.

AceMan

31.6 Rotazione

Adesso vediamo realmente come effettuare rotazioni. Utilizzando come sistema di riferimento un piano a 2 dimensioni è possibile convertire le coordinate cartesiane (x, y) di un punto in coordinate polari (r, t):

```
      _ V          V(x,y)=V'(r,t)            _ V'     r=distanza
  y|   /|                                  |   /|       punto-origine
   |  /            r=sqrt(x*x+y*y)         | r/
   | /             t=arctan(y/x)           | /        t=angolo compreso
   |/                                      |/) t        tra il vettore
   +-------->      x=r*cos(t)              +-------->    ed il semiasse
        x          y=r*sin(t)                            positivo x
```

Dopodiché per ruotare il punto intorno l'origine si dovrà sommare l'angolo di cui si vuol ruotare il punto alla variabile t e convertire le risultanti coordinate polari in cartesiane. Questo metodo risulta troppo lento per applicazioni in real time in quanto sono presenti quadrati, arcotangenti e radici quadrate; inoltre introducendo la variabile z si andrebbe incontro ad una pesante gestione delle coordinate polari (utilizzo di integrali tripli, ecc.): meglio trovare una soluzione più semplice e veloce. Immaginiamo di avere un vettore con ordinata nulla $V(x, 0)$ e vogliamo ruotarlo di un angolo a:

```
  y              vettore              y     _ Vr(xr,yr)   vettore
    |            coincidente            |   /|           ruotato
    |            l'ascissa              |  /             di a
    |                                   | /              radianti
    |    V(x,0)                         |/) a
    +----->->                           +------->
           x                                   x
```

Se volessimo trasformare in polari le coordinate di V risulterebbe $r = x$ $(x = sqr(x*x+0*0))$ e $t = 0$ $(0 = arctan(0/x)$. Per ruotare il vettore basta sommare a t l'angolo di rotazione a. Quindi:

```
(V   = generico vettore in coordinate cartesiane)
(V'  = generico vettore in coordinate polari)
(Vr  = vettore ruotato di a radianti in coordinate cartesiane)
(Vr' = vettore ruotato di a radianti in coordinate polari)
(Vxr = vettore ruotato con la sola componente x in coordinate cartesiane)
(Vyr = vettore ruotato con la sola componente y in coordinate cartesiane)

       V(x,y) = V'(r,t)
       Vr(xr,yr) = Vr'(r,t+a)    ->    xr=r*cos(t+a)  yr=r*sin(t+a)
Andiamo a sostituire xr e yr con le relative formule:
       Vr(r*cos(t+a),r*sin(t+a)) ->    r=x  t=0
Andiamo a sostituire r con x e t con 0:
       Vxr(x*cos(a),x*sin(a))
```

E questa è la formula per la rotazione di un vettore che non ha la componente y. Ora consideriamo un vettore che ha l'ascissa nulla $V(0, y)$:

```
       r=y    (=sqr(0*0+y*y))
       t=pi/2 (=arctan(y/0)=arctan(infinito))
       Vyr(y*cos(pi/2+a),y*sin(pi/2+a))
```

Un paio di formule trigonometriche ci dicono:

```
       cos(pi/2+a)=-sin(a)
       sin(pi/2+a)=cos(pi/2)
```

ma siccome utilizziamo un sistema di riferimento con l'asse y "rovesciato" dobbiamo cambiare un segno ad entrambe le formule per poterle sfruttare, che quindi diventano:

```
cos(pi/2+a)=sin(a)
sin(pi/2+a)=-cos(a)
```

Adesso la formula per ruotare un vettore senza componente x è uguale:

```
Vyr(y*sin(a),-y*cos(a))
```

Ma se guardiamo al caso generale, abbiamo un vettore V che ha entrambe le componenti x e y. Difatti un generico vettore con con le componenti x e y è uguale a:

```
V1(x,0)+V2(0,y)=V(x+0,0+y)=V(x,y)
```

Ora possiamo usare le formule di rotazione dei singoli casi per calcolare il caso generale con un'addizione tra vettori:

```
Vxr(x*cos(a)         ,x*sin(a)         ) +
Vyr(        +y*sin(a),        -y*cos(a)) =
-----------------------------------------
Vr (x*cos(a)+y*sin(a),x*sin(a)-y*cos(a))
```

Grazie a questa formula è possibile ruotare un qualsiasi vettore su uno spazio bidimensionale. In un ambiente 3D la formula appena descritta coincide con la rotazione intorno l'asse z (non c'è nessuna coordinata z cambiata). Per ruotare il punto intorno un altro asse basta lasciar fuori la relativa variabile e utilizzare le altre nella precedente espressione, il che si può sintetizzare con:

intorno l'asse z	intorno l'asse y	intorno l'asse x
xr=x*cos(a)+y*sin(a)	xr=x*cos(a)+z*sin(a)	yr=y*cos(a)+z*sin(a)
yr=x*sin(a)-y*cos(a)	zr=x*sin(a)-z*cos(a)	zr=y*sin(a)-z*cos(a)

31.7 Ottimizzazione delle rotazioni

Dato un angolo di rotazione per ogni asse (x, yez) con le precedenti formule sarebbero occorse ben 12 moltiplicazioni per poter ruotare un solo punto. Qui vedremo come effettuare rotazioni eseguendo 9 moltiplicazioni per ogni punto. Consideriamo:

```
ax = angolo di rotazione intorno l'asse x     s1 = sin(ax)  c1 = cos(ax)
ay = angolo di rotazione intorno l'asse y     s2 = sin(ay)  c2 = cos(ay)
az = angolo di rotazione intorno l'asse z     s3 = sin(az)  c3 = cos(az)
```

Ognuna delle variabili x, yez influenza le rotazioni intorno a due assi (nella rotazione intorno al proprio asse la variabile rimane inalterata), possiamo così indicare con $x'y'ez'$ le variabili parzialmente ruotate (cioè dopo la prima rotazione) e con $x''y''ez''$ le variabili completamente ruotate. Detto ciò le formule viste in precedenza corrispondono a:

```
x' = x*c1 + y*s1
y' = x*s1 - y*c1

x''= x'*c2 + z*s2   <-  coordinata x ruotata completamente
z' = x'*s2 - z*c2

y''= y'*c3 + z'*s3  <-  coordinata y ruotata completamente
z''= y'*s3 - z'*c3  <-  coordinata z ruotata completamente
```

che equivale a scrivere:

```
x''= (x*c1+y*s1)*c2+z*s2=c2*c1 *x + c2*s1 *y + s2 *z

y''= (x*s1-y*c1)*c3+((x*c1+y*s1)*s2-z*c2)*s3=
c3*s1 *x - c3*c1 *y + s3*s2*c1 *x + s3*s2*s1 *y - s3*c2 *z=
(s3*s2*c1+c3*s1) *x + (s3*s2*s1-c3*c1) *y + (-s3*c2) *z

z''= (x*s1-y*c1)*s3-((x*c1+y*s1)*s2-z*c2)*c3=
s3*s1 *x - s3*c1 *y - c3*s2*c1 *x - c3*s2*s1 *y + c3*c2 *z=
(-c3*s2*c1+s3*s1) *x + (-c3*s2*s1-c3*c1) *y + (c3*c2) *z

z''= (x*s1-y*c1)*s3-((x*c1+y*s1)*s2-z*c2)*c3=
     s3*s1 *x - s3*c1 *y - c3*s2*c1 *x - c3*s2*s1 *y + c3*c2 *z=

     (-c3*s2*c1+s3*s1) *x + (-c3*s2*s1-s3*c1) *y + (c3*c2) *z
     ^^^^^^^^^^^^^^^^^ zx   ^^^^^^^^^^^^^^^^^ zy   ^^^^^^^ zz
```

Dall'ultimo passaggio di ognuna di queste formule si può notare che non vengono calcolate coordinate parzialmente ruotate e che ogni coordinata ruotata equivale alla somma delle variabili (non ruotate) moltiplicate per un determinato fattore. Se precalcoliamo questi fattori potremo utilizzarli per tutti quei punti che devono essere ruotati nella stessa direzione. In questo modo avremo svolto semplicemente 9 moltiplicazioni per ogni punto (esclusi i precalcoli per i fattori). In sostanza dobbiamo calcolare prima queste costanti:

```
xx =  c2*c1
xy =  c2*s1
xz =  s2
yx =  c3*s1+s3*s2*c1
yy = -c3*c1+s3*s2*s1
yz = -s3*c2
zx =  s3*s1-c3*s2*c1 = s2*c1+c3*s1
zy = -s3*c1-s3*s2*s1 = c3*c1-s2*s1
zz =  c3*c2
```

poi per ogni punto si devono svolgere questi calcoli (sfruttando gli stessi fattori):

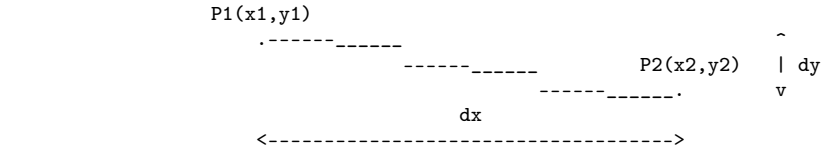
```
x'' = xx * x + xy * y + xz * z
y'' = yx * x + yy * y + yz * z
z'' = zx * x + zy * y + zz * z
```

Ed otterremo le tre coordinate ruotate.

Questo algoritmo risulterebbe meno efficiente del precedente se si utilizzassero pochi punti, mentre nel caso di una elevata quantità di vettori è possibile raggiungere notevoli risparmi in termini di calcolo.

31.8 Wireframe

Il wireframe è la più semplice ed antica tecnica atta a riprodurre poligoni. Consiste semplicemente nel tracciare linee che uniscano i vertici del poligono da rappresentare, nient'altro. Il tracciamento delle linee dovrà svolgersi sfruttando le coordinate proiettate dei punti (e non quelle con tre variabili xyz). Vediamo quindi come tracciare linee nel caso si stia programmando in un linguaggio che non permetta direttamente di svolgere questa funzione (come il C e l'Assembly). Analizziamo l'algoritmo di Bresenham. Abbiamo due punti $P1(x1, y1)$ e $P2(x2, y2)$ e vogliamo visualizzarne la linea che possa unirli:



consideriamo:

```
x2 > x1
y2 > y1
dx = x2-x1
dy = y2-y1
dx > dy
```

Tutti gli altri tipi di linee sono derivabili da questo tipo. Dopodiché andiamo a calcolare questi valori:

```
xl = x1            -> ascissa attuale del punto
yl = y1            -> ordinata attuale del punto
 d = 2*dx-dy       -> variabile di decisione
d1 = 2*dy          -> incremento di d (se d<0)
d2 = 2*(dy-dx)     -> incremento di d (se d=>0)
```

Finalmente vediamo l'algoritmo vero e proprio:

```
> loop di dx iterazioni
   > visualizza pixel alla posizione (xl,yl)
      > xl=xl+1
      > se d<0 allora:
      > d=d+d1
   > altrimenti:
      > d=d+d2
      > yl=yl+1
> prossima iterazione
```

Una linea è formata da un insieme di pixel, nel nostro caso il numero di pixel che compone la linea equivale a dx, quindi dobbiamo realizzare un loop che si ripete dx volte in cui per ogni iterazione bisogna visualizzare un punto. Ma quali coordinate dovrà avere questo punto? Indichiamo con xl e yl le coordinate del punto da proiettare su video, le quali inizialmente coincideranno con quelle di $P1(x1, y1)$. Al termine di ogni iterazione andiamo ad incrementare xa, in questo modo uscendo dal ciclo xa coinciderà con $x2$ (perché $x2 = x1 + dx$). Cosa succede invece alla ordinata del pixel? La incrementiamo semplicemente quando la variabile d è positiva. È possibile utilizzare anche un altro algoritmo per tracciare linee, il quale risulta spesso più efficiente di quello di Bresenham (soprattutto se realizzato in Assembly) che sfrutta il principio di interpolazione lineare, lo vedremo più dettagliatamente nel paragrafo relativo al fill e scan line.

31.9 Hidden Face

Hidden face significa faccia nascosta, in questo paragrafo vedremo come eliminarla. Difatti nella realtà in un solido non trasparente non sono visibili tutte le facce per ovvi motivi. Visualizzare sul monitor solo le facce visibili è sicuramente più realistico rispetto a tracciarle tutte.

Il più semplice e intuitivo algoritmo è quello del "pittore". Consiste nell'ordinare le facce che compongono l'oggetto in base alla componente z. In seguito si devono disegnare le facce partendo da quella più lontana sino a quella più vicina; in questo modo le facce disegnate per ultime

saranno visibili, mentre sulle nascoste verranno disegnate quelle visibili. A suo discapito questo algoritmo include un pesante spreco di tempo macchina, inoltre è praticamente inutilizzabile per grafica in wireframe, però permette a qualsiasi oggetto (che non sia wireframe) di essere visualizzato correttamente.

Un'altra via consiste nel calcolare la normale (la retta perpendicolare) su ogni faccia, controllare se punta verso l'osservatore ed in caso contrario non visualizzarla. Questo algoritmo è valido solo se i vertici che delimitano la faccia sono posti in memoria in senso orario, in quanto i calcoli che vedremo sfruttano questa caratteristica. Inoltre gli oggetti dovranno necessariamente essere convessi, ossia non devono esserci facce che possano "oscurare" (non nascondere!) altre facce. La retta normale la possiamo considerare una grandezza vettoriale che nello spazio viene indicata come un comune vettore con tre componenti.

La visibilità di un poligono dipende esclusivamente dalla sua orientazione lungo l'asse z. Essendo più precisi possiamo dire che la sola componente z è necessaria per sapere se la faccia è nascosta o meno. Consideriamo la nostra faccia come la seguente:

```
                  per individuare un piano bastano tre punti. Di
   A(x1,y1)       conseguenza, per ricavare la normale sul piano
    /\            coincidente la nostra faccia, saranno sufficienti
   /  \           le coordinate dei primi tre vertici della faccia.
D /    \ B(x2,y2) Lasciando da parte eventuali dimostrazioni, è
  \    /          possibile affermare che la componente z della
   \  /           normale è  equivalente a:
    \/
   C(x3,y3)         (x2-x1)*(y3-y1)-(x3-x1)*(y2-y1)
```

Se il risultato è minore o uguale a zero la faccia è nascosta, altrimenti è visibile. Per sapere semplicemente se questa componente z è maggiore o minore di zero potremmo anche utilizzare le coordinate proiettate, ovvero:

```
(xp2-xp1)*(yp3-yp1)-(xp3-xp1)*(yp2-yp1)
```

Questo è quanto basta per sapere se un poligono è o meno visibile.

31.10 Algoritmo del pittore

L'algoritmo del pittore consiste semplicemente nel visualizzare un oggetto (o una scena) 3D tracciando i poligoni partendo da quello più distante verso quello più vicino al punto di vista. Il suo scopo è di risolvere il problema causato dal tracciamento delle facce parzialmente oscurate da altre (ovvero visualizzare oggetti concavi).

Questo semplice principio è lo stesso che un qualsiasi pittore utilizza quando deve dipingere un quadro. Difatti si inizia sempre col disegnare (ad esempio) i monti (che si trovano più distanti dal punto di vista) sino a disegnare gli elementi più vicini al pittore, quali possono essere un ruscello o una persona.

Per realizzare l'algoritmo del pittore è sufficiente utilizzare un array monodimensionale a cui ogni coppia di posizioni corrispondano il puntatore o l'indice di ogni faccia e la componente z di quella faccia. Consideriamo di poter numerare le facce del nostro oggetto, quindi di poter indicare una singola faccia con un determinato numero. Il numero di elementi necessari al questo buffer sono equivalenti a:

```
Dimensione Buffer = 1 + (numero facce oggetto)*2
```

Nella prima posizione viene salvato il numero di facce da tracciare. Invece nelle successive posizioni vengono salvati (per ogni poligono) il numero identificatore della faccia (che può essere un indice o un puntatore e la componente z di quella faccia di cui abbiamo appena parlato. La coordinata z relativa alla faccia è equivalente alla media aritmetica delle componenti z dei vertici di quel poligono.

Successivamente dovremo ordinare in ordine crescente gli elementi di tale struttura in base alla coordinata z. Una volta ordinato l'array tracceremo i poligoni a video nella successione definita dagli identificatori delle facce presenti nel nostro array già ordinato (partendo dallo specificatore con la z maggiore sino ad arrivare allo specificatore della faccia con la z minore); così facendo saremo sicuri di visualizzare le facce dalle più lontane alle più vicine; ciò ci permette di rappresentare correttamente sullo schermo anche oggetti concavi.

Per snellire questo buffer possiamo calcolare la componente z della normale di ogni faccia (Nz) e, solamente se Nz risulterà positiva, salveremo l'identificatore e la z di quella faccia; in caso contrario (se $Nz < 0$) passiamo direttamente alla prossima faccia. In altre parole andiamo a considerare solo le facce orientate verso l'osservatore (consultare il paragrafo relativo all'hidden face per ulteriori approfondimenti).

Se applicato nella maniera dovuta (ovvero utilizzando un algoritmo di ordinamento ottimizzato), l'algoritmo del pittore può considerarsi uno dei metodi più veloci nella rimozione delle facce nascoste durante il tracciamento di un oggetto o di una scena 3D. I suoi principali svantaggi risiedono nella difficoltà di implementazione nel caso di intersezioni tra poligoni; nell'imprecisione visiva in certi mondi 3D complessi e nell'impossibilità di tracciare correttamente i poligoni in alcuni casi (si immagini di voler visualizzare una scena in cui vi siano un enorme poligono che rappresenti un pavimento su cui giace un piccolo cubo. Poniamo caso caso che la componente z del centro di tale cubo corrisponda con quella del poligono coincidente il pavimento. Dopo aver tracciato i poligoni su schermo può accadere che alcune facce del cubo vengano disegnate prima del pavimento, quindi alcuni poligoni, che dovrebbero essere visibili, non verranno visualizzati su schermo).

31.11 Filled vector e scan-line

I filled vector non rappresentano altro che poligoni "riempiti" di un determinato colore. Realizzare una routine di fill significa colorare il contenuto di un poligono conoscendone le coordinate proiettate dei vertici. Immaginiamo che il poligono da riempire sia il seguente:

```
|        A|\            potremmo procedere nella seguente maniera:
|         | \           a partire dalla coordinata y minore del poligono
|         |  \          (in questo caso quella di A) e via via arrivando
|         |   \         all'ultima posizione verticale (ossia y di D)
|         |    \        coloriamo la riga delimitata dalle posizioni x
|         |     \ B     dei lati in quella posizione y.
|        D \     |
|           \    |      Il principio del fill si basa sul riempimento di
|            \   |      righe orizzontali partendo dalla riga superiore
|             \  |      del poligono sino a quella inferiore.
|y             \ |
v               \|C     Vediamo un esempio relativo ad una sola riga:

|        A|\
|         | \           dobbiamo riempire la riga indicata nella figura.
|         |  \          Partiamo dalla coordinata x del lato AD.
|  riga   |   \         Iniziamo col colorare questo punto.
|   da -->|****\        Passiamo al punto successivo (ossia quello a
| riempire|     \ B     destra) e coloriamo anch'esso; procediamo col
```

```
|        D \     |      colorare i pixel successivi fin quando arriviamo
|           \    |      a colorare il punto posto sul lato AB e passiamo
|            \   |      alla prossima riga.
|             \  |      Ciò significa che per ogni riga ci servono le
|y             \ |      coordinate x del punto estremo a destra e del
v              \|C     punto estremo a sinistra.
```

Ora che abbiamo capito cosa fare vediamo come realizzare il tutto. Bisogna utilizzare due tabelle (matrici unidimensionali) in memoria di dimensioni equivalenti al numero di pixel verticali rappresentabili sullo schermo (es.: in una risoluzione 320*200 dobbiamo avere due tabelle di 200 valori ciascuna). Consideriamo ogni posizione delle tabelle una posizione y nello schermo ed il contenuto del primo array come la corrispondente componente x del punto estremo a sinistra mentre il valore del secondo array come la componente x del punto estremo a destra. Così facendo basterà colorare tutti i pixel alla riga corrispondente la posizione delle tabelle partendo dalla posizione x contenuta dalla prima tabella sino alla posizione x contenuta dalla seconda (tutti i punti di una riga hanno la stessa ordinata). Facciamo un esempio pratico:

```
 0| x=0 ->A. <- x=0        per semplicità  consideriamo i segmenti AB e CD
 1| x=0 -> .. <- x=1       inclinati a 45 gradi . I vertici sono:
 2| x=0 -> . . <- x=2      A(0,0)  B(5,5)  C(5,10)  D(0,5)
 3| x=0 -> .  . <- x=3     le nostre due tabelle saranno:
 4| x=0 -> .   . <- x=4    +-------------------------------------------+
 5| x=0 ->D.    .B<- x=5   |TAB1| 0| 0| 0| 0| 0| 0| 1| 2| 3| 4| 5|..|..|
 6|  x=1 -> .   . <- x=5   +-------------------------------------------+
 7|   x=2 -> .  . <- x=5   |TAB2| 0| 1| 2| 3| 4| 5| 5| 5| 5| 5| 5|..|..|
 8|    x=3 -> . . <- x=5   +-------------------------------------------+
 9|     x=4 -> .. <- x=5   Per riempire il poligono basterà  colorare i
10|y     x=5 -> .C<- x=5   pixel compresi tra i corrispondenti valori
  v                        delle due tabelle utilizzando come componente
                           y l'indice delle tabelle (che è  lo stesso per
entrambe). A volte è  possibile eliminare i due valori estremi delle
tabelle, quando in quelle righe vi è  un solo pixel (come nel nostro
esempio). Adesso non ci rimane altro che ricavarci il contenuto di questi
due array.
```

Per definizione si indica col termine "scanline" una linea orizzontale su schermo del poligono. Un "edge" non è altro che uno dei due pixel ai bordi della scanline, quindi gli array di cui dovremo ricavarci il contenuto possiamo chiamarli "edge buffer". Le tabelle contengono semplicemente le coordinate x di tutti i punti che compongono i lati del poligono; inoltre queste ascisse sono ordinate in base alla loro componente y. In pratica dobbiamo svolgere una routine di tracciamento di linee per tutti i lati della faccia, in cui non visualizziamo i pixel, bensì salviamo la componente x in un array la cui posizione equivale alla y di quel punto. Questa procedura è denominata "scan conversion" e praticamente consiste nel suddividere il poligono in un insieme di righe e colonne. Per realizzare la scan conversion è possibile utilizzare l'algoritmo di Bresenham, però conviene sfruttare il procedimento di interpolazione lineare, che risulta più efficiente. Vediamo sinteticamente in cosa consiste. Consideriamo due generici punti $A(x1, y1)$ e $B(x2, y2)$ dove $y2 > y1$. Ora calcoliamo:

```
dx = x2 - x1     <-- lunghezza della linea che unisce A e B
dy = y2 - y1     <-- altezza della linea che unisce A e B
stepx = dx / dy  <-- numero di pixel orizzontali su ogni riga
```

Mentre l'algoritmo generale è :

```
> x = x1
> y = y1
```

```
> loop di dy iterazioni
   > se è  libera la posizione y della tab1:
      > salva x in posizione y della tab1
   > altrimenti:
      > salva x in posizione y della tab2
   > x = x + stepx
   > y = y + 1
> prossima iterazione
```

Questo algoritmo permette di riempire parzialmente un edge buffer nel caso $y2 > y1$, se invece risulta $y1 > y2$ basterà scambiare entrambe le coordinate dei due punti (ossia considerare $y1$ come $y2$ e $x1$ come $x2$). Per riempire completamente l'edge buffer basterà ripetere tale procedura per ogni lato del poligono. La tab1 è l'array che contiene i punti estremi a sinistra mentre la tab2 punti estremi a destra. Col nostro algoritmo può capitare che alcuni i valori scritti nella tab1 appartengano alla tab2, e viceversa. Ciò significa che le coordinate x presenti nella tab1 potranno essere gli edge a destra mentre i valori presenti nella tab2 potranno essere gli edge a sinistra. Vediamo cosa fare per evitare questo inconveniente.

Se abbiamo i punti salvati in senso orario il tutto risulta più semplice e veloce. Dati due punti $A(x1, y1)$ e $B(x2, y2)$ posti in senso orario, se $y1$ è maggiore di $y2$ allora l'insieme dei punti che formano quel lato appartiene alla tab1 (quella contenente le posizioni x minori), in caso contrario quei punti apparterranno alla tab2 (contenente le x maggiori). Ecco l'algoritmo completo per la scan conversion relativo ad un singolo lato:

```
> confronta y1 con y2
   > se y1=y2:
     > esci dalla procedura!
   > se y1>y2:
     > la giusta tab è  tab1
   > se y1<y2:
     > la giusta tab è  tab2
     > scambia y1 con y2
     > scambia x1 con x2
> dy = y1 - y2
> dx = x1 - x2
> stepx = dx / dy
> x = x2
> y = y2
> loop di dy iterazioni
   > salva x in posizione y nella giusta tab
   > x = x + stepx
   > y = y + 1
> prossima iterazione
```

Una volta eseguita la scan conversion del poligono dovremo calcolare la componente y minore dei quattro punti che compongono i vertici del poligono ed l'altezza in pixel del poligono stesso. La coordinata y minore rappresenta l'indice delle tabelle da cui partire per riempire il poligono e quindi la posizione y superiore del poligono. L'altezza del poligono è equivalente alla differenza tra la y maggiore e la y minore e ci serve per sapere quante righe dovremo riempire per l'attuale poligono.

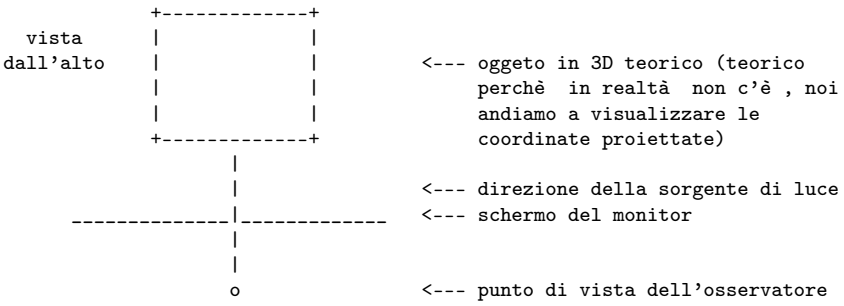
Riassumendo, per effettuare il fill di un poligono si devono svolgere i seguenti passi:

- definire in memoria due tabelle dimensionate ad ys valori (dove ys rappresenta l'altezza in pixel dello schermo);
- calcolare la y minore dei vertici e l'altezza del poligono;

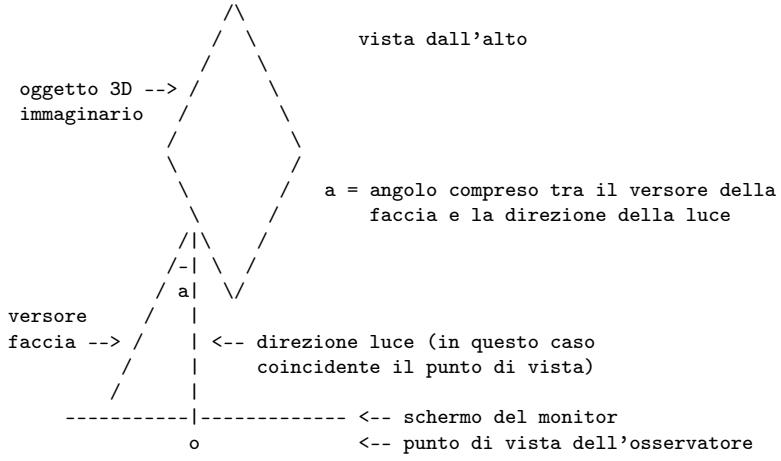
- svolgere la scan conversion del poligono salvando le coordinate x iniziali e finali (edge) di ogni scanline nell'edge buffer;
- partendo dalla posizione y minore, riempire la riga delimitata dalle posizioni x contenute dall'edge buffer stesso per un numero di volte equivalente all'altezza del poligono.

31.12 Flat shading

Siamo arrivati all'analisi del primo (e più semplice) algoritmo di shading, grazie al quale potremo attribuire ad ogni poligono comprendente l'oggetto una precisa intensità di luce, la quale verrà determinata in base all'orientazione della faccia rispetto alla sorgente di luce. Il flat shading permette di attribuire ad ogni faccia un solo colore che determinerà quanto il poligono sia illuminato. Facciamo un esempio:



Consideriamo che la sorgente di luce corrisponda con il punto di vista, la sua direzione è perpendicolare lo schermo. Definiamo l'angolo di inclinazione della faccia rispetto la sorgente di luce come l'angolo compreso tra la retta corrispondente la direzione della luce ed il versore della faccia (ovvero la retta normale). Più questo angolo è piccolo e più il poligono risulta orientato verso l'osservatore. Possiamo intuire che tanto la faccia è posta di fronte l'osservatore e tanto sarà maggiore l'intensità di luce applicata su quella faccia. Di conseguenza ad un angolo minore corrisponderà una luminosità maggiore della faccia. Ecco un altro esempio:



L'intensità di luce attribuibile al poligono è proporzionale al coseno di questo angolo. Sappiamo che il generico risultato di $\cos(a)$ è compreso tra -1 e 1. Da notare che se il poligono

è visibile il nostro angolo varia da 0 a 90 gradi, altrimenti la faccia risulta nascosta (è possibile utilizzare questa caratteristica per l'eliminazione dell'hidden face!). Quindi il valore del coseno relativo al nostro angolo copre un range di valori da 0 e 1. Inoltre, utilizzando 256 colori, basterà moltiplicare il coseno per 256 (oppure effettuare uno shift di 8 bit a sinistra) e otterremo il pixel chunky col quale dovremo riempire la relativa faccia! Vediamo come ricavare questo valore.

Innanzitutto dobbiamo specificare la palette dei colori da utilizzare. Ciò è possibile definendo nella memoria video una tavolozza che parte dal colore di luminosità minore sino ad arrivare gradualmente al colore più chiaro. Per il calcolo del coseno conviene sfruttare la regola di Lambert, la quale afferma che il prodotto scalare tra due rette espresse come grandezze vettoriali è equivalente al prodotto della lunghezza dei relativi vettori e del coseno dell'angolo limitato dalle rette stesse, ovvero l'angolo a. Quindi, per conoscere $\cos(a)$, non dovremo far altro che svolgere questo prodotto scalare e dividere il risultato per il prodotto delle lunghezze dei due vettori.

Per calcolare il prodotto scalare di due vettori si esegue il prodotto delle corrispondenti componenti e poi si sommano i risultati, ad esempio:

```
H=(xh,yh,zh)  ;  K=(xk,yk,zk)
H*K=xh*xk+yh*yk+zh*zk         <-- prodotto scalare
```

Per ricavare una lunghezza di un vettore possiamo ricorrere al teorema di Pitagora grazie al quale si può affermare che la lunghezza è equivalente alla radice quadrata della somma dei quadrati di ogni componente.

Verifichiamo come calcolare i coefficienti x, y e z del versore della faccia:

```
Nx=(y2-y1)*(z3-z1)-(y3-y1)*(z2-z1)    <-+--- i tre coefficienti
Ny=(z2-z1)*(x3-x1)-(z3-z1)*(x2-x1)    <-|    della retta normale
Nz=(x2-x1)*(y3-y1)-(x3-x1)*(y2-y1)    <-+
```

N.B.: i punti devono essere posti in memoria in senso orario!

```
x1, y1, z1 = componenti del primo punto del poligono
x1, y2, z2 = componenti del secondo punto del poligono
x3, y3, z3 = componenti del terzo punto del poligono
```

Infine ecco la formula per calcolare il pixel chunky:

```
                    Nx*lx + Ny*ly + Nz*lz
cos(a)=-------------------------------------------------
       sqrt(Nx*Nx+Ny*Ny+Nz*Nz) * sqrt(lx*lx+ly*ly+lz*lz)

pixel chunky = 256*cos(a)

 a = angolo tra il versore e la direzione della sorgente di luce
lx = componente x della sorgente di luce
ly = componente y della sorgente di luce
lz = componente z della sorgente di luce
```

Le coordinate lx, ly e lz rappresentano la posizione della sorgente di luce. Nel caso la luce coincida con il punto di vista dell'osservatore, le relative coordinate risulteranno:

```
lx=0  ;  ly=0  ;  lz=-256
```

zl è uguale all'opposto della distanza tra l'osservatore e lo schermo (nel nostro caso la distanza tra l'osservatore e lo schermo è 256).

31.13 Ottimizzazioni per il calcolo della sorgente di luce

In questa parte vediamo come velocizzare il nostro motore 3D nel caso comprenda l'implementazione di una sorgente di luce reale.

Una prima ottimizzazione consiste nell'utilizzare un buffer dove andremo a precalcolare tutte le normali di ogni faccia (o di ogni vertice nel caso del gouraud shading); poi anziché calcolare ad ogni frame tutte le normali, ruotiamo i nostri versori precalcolati dello stesso angolo con cui ruotiamo i vertici dell'oggetto sfruttando la stessa identica procedura descritta in precedenza (utilizzando preferibilmente l'algoritmo a 9 moltiplicazioni).

Abbiamo detto che il prodotto scalare tra il vettore normale ed il vettore corrispondente la sorgente di luce moltiplicato 256 ci permette di conoscere il pixel chunky. Ora vediamo come eliminare subito le radici quadrate e la divisione. Analizziamo di nuovo la formula:

```
                    Nx*lx + Ny*ly + Nz*lz
cos(a)=-------------------------------------------------
       sqrt(Nx*Nx+Ny*Ny+Nz*Nz) * sqrt(lx*lx+ly*ly+lz*lz)
```

Per attuare questa ottimizzazione dobbiamo rendere unitario il vettore normale ed il vettore corrispondente la sorgente di luce. Rendere unitario un vettore significa dividere ognuna delle componenti per la sua distanza dall'origine; ciò fa sì che le nuove componenti abbiano un range compreso tra -1 e $+1$, ecco perché si dice unitario. Vediamo algebricamente come rendere unitario un qualsiasi versore:

```
                  Nx
uNx=---------------------------
    sqrt(Nx*Nx + Ny*Ny + Nz*Nz)

                  Ny
uNy=---------------------------
    sqrt(Nx*Nx + Ny*Ny + Nz*Nz)

                  Nz
uNz=---------------------------
    sqrt(Nx*Nx + Ny*Ny + Nz*Nz)
```

Più genericamente, dato un vettore $V(x, y, z)$, per calcolare le componenti del relativo vettore unitario $uV(ux, uy, uz)$:

```
                  x
ux=--------------------
   sqrt(x*x + y*y + z*z)

                  y
uy=--------------------
   sqrt(x*x + y*y + z*z)

                  z
uz=--------------------
   sqrt(x*x + y*y + z*z)
```

Per rendere unitaria la sorgente di luce possiamo anche evitare di dividere ogni sua componente per la lunghezza della medesima, infatti siamo noi a decidere dove si trova, quindi arbitrariamente possiamo assegnare coordinate unitarie. Ad esempio tornando al caso che la luce coincida col punto di vista:

```
ulx=0  ;  uly=0  ;  ulz=-1
```

Quindi la formula per calcolare l'intensità di luce viene ridotta a:

```
cos(a) = uNx*ulx + uNy*uly + uNz*ulz
pixel chunky = 256*cos(a)
```

Rendere unitario un vettore e poi ruotarlo o ruotare un vettore e poi renderlo unitario significa svolgere la stessa funzione; quindi al momento in cui andiamo a precalcolare le normali possiamo renderle subito unitarie, in seguito andremo a ruotare i versori già resi unitari. In questo modo evitiamo di eseguire 2 radici quadrate e una divisione per vertice ad ogni frame!

N.B.: nel caso si voglia muovere la sorgente di luce, utilizzando questa ottimizzazione non si potranno attuare traslazioni della sorgente di luce, ma solo rotazioni, in quanto la distanza origine-luce deve restare costante.

L'ultima ottimizzazione consiste nel mantenere fissa in un punto la sorgente di luce, più precisamente diciamo che deve coincidere sempre con il punto di vista dell'osservatore. Naturalmente utilizzeremo coordinate unitarie per i versori e la luce. Vediamo la formula per calcolare il pixel chunky in questo particolare caso:

```
pixel chunky = 256*( uNx*ulx + uNy*uly + uNz*ulz) =
             = 256*( uNx*0   + uNy*0   + uNz*(-1))=
             = -256*uNz
```

Ora il nostro pixel chunky dipende solo da uNz, quindi possiamo precalcolare per ogni vertice $-256 * uNz$ al posto del semplice uNz, ruotarlo e utilizzare subito questo valore come pixel chunky. In questo modo evitiamo 3 moltplicazioni e 2 addizioni. Inoltre siccome ci serve solo uNz possiamo benissimo evitare di ruotare uNx e uNy, risparmiando la bellezza di altre 6 moltiplicazioni per faccia (o per vertice nel caso del gouraud). In totale risparmiamo ben 9 moltiplicazioni e 2 addizioni per faccia (o per vertice nel gouraud)! Naturalmente dovremo precalcolare oltre a $-256 * uNz$ anche uNx e uNy moltiplicati 256 ($256 * uNx$, $256 * uNy$) che servono per poter ruotare $-256 * uNz$. Inoltre se invertiamo la nostra palette potremo utilizzare $256 * uNz$ al posto di $-256 * uNz$.

31.14 Gouraud shading

Questo algoritmo di ombreggiatura permette di sfumare l'interno di ogni poligono al contrario del flat shading col quale si assegna un unico colore per faccia.

Innanzitutto bisogna calcolare il versore di ogni vertice dell'oggetto anziché di ogni poligono. Le componenti della normale sul vertice sono equivalenti alla media aritmetica delle componenti delle normali di tutte le facce che toccano quel vertice. Facciamo un esempio:

```
 ____          sia V un generico vertice di un cubo appartenente
 /f2 /\        alle facce f1, f2 e f3. Consideriamo le normali di
/___/V \       codeste facce, chiamiamo questi versori N1,N2,N3.
\f1 \f3/
 \___\/        N1(Nx1,Ny1,Nz1)  N2(Nx2,Ny2,Nz2)  N3(Nx3,Ny3,Nz3)
```

Allora la normale su V è equivalente a:

```
NV( (Nx1+Nx2+Nx3)/3, (Ny1+Ny2+Ny3)/3, (Nz1+Nz2+Nz3)/3 )
```

In questo caso le facce appartenenti a V sono 3, a seconda dell'oggetto che si vuol utilizzare il numero di poligoni appartenenti ad un vertice cambiano.

Una volta precalcolate tutte le normali (preferibilmente già unitarie) su ogni spigolo dovremo calcolare la quantità di luce che cade su ognuno dei vertici, ovvero il pixel chunky, utilizzando la legge già studiata nel flat shading (magari sfruttando eventuali ottimizzazioni citate nel precedente paragrafo).

In seguito facciamo la scan conversion (spiegata nel paragrafo dedicato al fill e scan line) di tutti i poligoni visibili.

Adesso dobbiamo interpolare linearmente per ogni faccia i pixel chunky appartenenti ai vertici di quella faccia. In pratica dovremo svolgere una semplice scan conversion del poligono utilizzando i pixel chunky al posto delle coordinate x dei vertici, tutto qui. Naturalmente ciò va svolto solamente se la faccia risulta visibile.

Non rimane altro che effettuare il fill vero e proprio dei poligoni. Come per un normale fill bisogna eseguire un loop ad iterazioni equivalenti all'altezza in pixel del poligono. Ad ogni iterazione preleviamo di volta in volta le coordinate x iniziali e finali dalle tabelle delle scan line (come per un normale fill), però stavolta preleviamo anche i pixel chunky iniziali e finali. Ora dobbiamo interpolare il pixel chunky iniziale con quello finale partendo dalla coordinata x iniziale sino ad arrivare a quella finale. Per far ciò basta utilizzare l'algoritmo per tracciare una scan line con le seguenti modifiche:

- utilizzare il pixel chunky iniziale al posto della coordinata $x1$;
- utilizzare il pixel chunky finale al posto della coordinata $x2$;
- utilizzare la x iniziale al posto della coordinata $y1$;
- utilizzare la x finale al posto della coordinata $y2$;
- utilizzare la riga dello schermo chunky da "fillare" come tabella dove la scan line verrà salvata.

Ed ecco realizzato il gouraud shading!

31.15 Phong shading

Il phong shading permette di assegnare ad ogni pixel la sua reale intensità di luce, al contrario del gouraud nel quale si generano sfumature all'interno di ogni faccia tra le intensità di luce di ogni vertice dell'oggetto. La maggiore definizione che si ottiene col phong rispetto al gouraud comporta allo stesso tempo un drastico aumento delle operazioni che il processore deve svolgere. La pesantezza dei calcoli da eseguire è tale da impedire agli attuali elaboratori di tracciare soddisfacenti scene in phong shading in tempo reale.

Nel gouraud calcoliamo la reale intensità di luce su ogni vertice, dopodiché ogni colore viene interpolato lungo ogni lato del poligono, infine si interpolano i colori posti sui lati estremi a sinistra con i colori posti sui lati estremi a destra in modo da riempire l'intero poligono. Nel phong invece interpoliamo sempre le normali, non si interpolano mai i colori. Una volta determinati i versori su ogni vertice, questi devono essere interpolati lungo ogni lato; successivamente i versori posti lungo i lati estremi a sinistra verranno interpolati con quelli posti lungo i lati estremi a destra, quindi per ogni singolo verrà calcolato il colore sfruttando la tradizionale formula più volte studiata.

Il phong ci impedisce di sfruttare le diverse ottimizzazioni possibili col gouraud shading e col flat shading. Infatti nel phong è impossibile utilizzare normali unitarie in quanto nel momento in cui queste vengono interpolate la loro lunghezza (equivalente al risultato dell'espressione $sqrt(Nx*Nx+Ny*Ny+Nz*Nz))$ può variare. Quindi occorre eseguire almeno una divisione ed una radice quadrata per pixel, il che non è poco.

31.16 Reflection mapping

Se un solido riflette sempre e solo una singola immagine (in gergo denominata "texture") allora possiamo affermare che su quel solido è stato applicato il reflection mapping. Nel caso la texture corrisponda approssimativamente alla rappresentazione bidimensionale di una luce (es.: un cerchio il cui centro risulta molto chiaro mentre agli orli viene sfumato in una tinta più scura), è possibile raggiungere effetti simili (e a volte superiori) al phong e al gouraud.

Spesso questo effetto viene erroneamente confuso con l'environment mapping, il quale permette invece di riflettere un intero ambiente che circonda l'oggetto (il quale ambiente è spesso definito per semplicità come un cubo, quindi in questo caso sul solido verranno riflesse sei immagini). Tuttavia è possibile considerare il reflection mapping come un'approssimazione dell'environment mapping.

In questo paragrafo verrà descritto come realizzare il reflection mapping utilizzando esclusivamente texture di dimensioni 256*256 pixel. L'implementazione di texture di dimensioni differenti è facilmente derivabile.

Vediamo dettagliatamente come realizzare un comune oggetto in reflection mapping. Inizialmente ci andiamo a precalcolare tutti i versori unitari su ogni vertice (come per il gouraud) e li moltiplichiamo per 128 (oppure applichiamo un semplice scorrimento di 7 bit a sinistra), il che matematicamente si traduce in:

```
    _                                          _
   |  PVx = 128*Nx / sqrt(Nx*Nx + Ny*Ny + Nz*Nz)  |
PV |  PVy = 128*Ny / sqrt(Nx*Nx + Ny*Ny + Nz*Nz)  |
   |_ PVz = 128*Nz / sqrt(Nx*Nx + Ny*Ny + Nz*Nz) _|
```

Chiamiamo PV il vettore che ha come componenti questi 3 valori. La normale unitaria ha come coordinate 3 valori che comprendono numeri reali tra -1 e $+1$. Ora abbiamo PVx, PVy e PVz che rispetto ai versori unitari sono moltiplicati 128, questo vuol dire che copriranno un raggio di valori compresi tra -128 e +128 (anche se in realtà tali valori non superano mai +127). E qui finisce la fase di precalcolo.

In tempo reale dobbiamo ruotare il vettore PV per ogni vertice (casomai sfruttando gli stessi fattori di rotazione dei punti se si ha realizzato la rotazione a 9 moltiplicazioni). Del vettore PV ci servono solo le componenti x e y ruotate, quindi possiamo anche non ruotare PVz evitando così di svolgere almeno 3 moltiplicazioni e 2 addizioni per vertice (naturalmente è necessario precalcolare PVz per ogni vertice per poter ruotare PVx e PVy). Ad ognuna delle componenti ruotate x e y del vettore PV addizioniamo il valore 128. Al termine di questi calcoli, il range dei valori che PVx e PVy potranno coprire sarà compreso tra 0 e 255.

In realtà $((PVxruotato) + 128)$ e $((PVyruotato) + 128)$ rappresentano le coordinate della texture da mappare (ossia tracciare) sul poligono. Ciò significa che se abbiamo un poligono delimitato da 4 punti, dobbiamo mappare su quel poligono la parte di texture delimitata dai 4 relativi PVx e PVy ruotati (e sommati con 128). Quindi basterà mappare il "pezzo" di texture su quel poligono e ripetere il tutto per ogni faccia visibile per realizzare il reflection mapping!

Ora vediamo come tracciare la parte di texture una volta calcolati i nuovi PVx e PVy. Innanzitutto dobbiamo svolgere la scan conversion del poligono, in più bisogna interpolare PVx e PVy lungo tutti i lati della nostra faccia, il che significa fare 2 ulteriori scan conversion del poligono utilizzando i PVx e i PVy al posto delle coordinate x dei vertici. Quindi in tutto vi sono 3 scan conversion: la prima è quella tradizionale, la seconda viene fatta sostituendo PVx alle x dei vertici, mentre la terza utilizza i PVy al posto delle x (facciamo esattamente come per le normali nel phong, con la differenza che consideriamo 2 componenti (PVx e PVy) al posto di 3 $(Nx, NyeNz)$). Abbiamo eseguito la scan conversion del poligono, quel di cui abbiamo bisogno

è un algoritmo che permetta di associare ad ogni punto appartenente alla faccia un determinato pixel della texture. Consideriamo la seguente figura come la nostra faccia proiettata a video:

```
            .                 Dopo aver interpolato PVx e PVy e svolto
           . .                la scan conversion, per ogni coppia di
          .   .               punti sulla stessa posizione y dello
    P1 -> .     . <- P2       schermo, ad esempio P1 e P2, conosciamo
         .       .            la loro coordinata x assieme a PVx e PVy.
          .     .             Adesso dobbiamo interpolare PVx e PVy
           .   .              dal punto P1 al punto P2, in questo modo
            . .               sapremo il valore di PVx e PVy per tutti
             .                i punti del poligono. Per interpolare
                              questi 2 valori lungo una riga si deve
P1 -> x1, y, PVx1, PVy1       applicare l'algoritmo generale per il
P2 -> x2, y, PVx2, PVy2       tracciamento di una scan line utilizzando
dPVx = PVx1-PVx2              dx al posto di dy,  dPVx (per interpolare
dPVy = PVy1-PVy2              PVx) e dPVy (per interpolare PVy) al posto
  dx = x1-x2                  di dx, proprio come abbiamo fatto nel
                              gouraud per interpolare i pixel chunky.
```

Come già accennato, PVx e PVy rappresentano le coordinate del pixel texture da tracciare. Appena svolte le 3 scan conversion conoscevamo PVx e PVy appartenenti ad ogni vertice della faccia. Quindi, interpolando PVx e PVy lungo tutto il poligono, ricaveremo le coordinate dei punti della texture per tutti i punti della faccia! Ora, con delle semplici operazioni di copia, potremo mappare il poligono associando, ad ogni suo punto, il pixel chunky della texture in posizione (PVx, PVy).

31.17 Affine texture mapping

Questo effetto permette il tracciamento di un'intera immagine su di un poligono, in pratica è come se "incollassimo" su ogni faccia una texture (ovvero una semplice immagine chunky posta in memoria). In questo paragrafo tratteremo del texture mapping senza prospettiva (il cosiddetto "affine texture mapping"), il quale risulta essere uno degli algoritmi più veloci per mappare un'immagine su di un poligono, ma allo stesso tempo anche il meno realistico.

Per completezza definiamo gli assi relativi alla texture (posta in memoria, non quella visualizzata su schermo) hanno come origine il punto più in alto a sinistra. L'ascissa di tale sistema viene denominata $< u >$ mentre l'ordinata $< v >$; quindi un generico punto della texture può essere indicato come $P(u, v)$. Da notare che le componenti u e v non possono assumere valori negativi.

Consideriamo di utilizzare poligoni formati da 4 lati, ogni spigolo del poligono coincide con uno spigolo della faccia, ciò che dobbiamo fare è tracciare tutta la texture sul poligono. In pratica è come se dovessimo realizzare il reflection mapping sapendo che le coordinate della texture da mappare sono sempre costanti e coincidono esattamente con i 4 vertici della texture stessa. Se la texture è 256*256 pixel realizzeremo un algoritmo di reflection mapping sapendo che $PV1(0, 0)$, $PV2(255, 0)$, $PV3(255, 255)$, $PV4(0, 255)$ sono gli stessi per ogni poligono. In altre parole andiamo ad interpolare le coordinate x e y della texture (ovvero u e v) lungo tutto il poligono, in modo tale da sapere per ogni pixel appartenente alla faccia da tracciare quale sia il relativo punto della texture. Tutto qui.

Per chiarire meglio le idee osserviamo un semplice esempio:

```
A +-----------------+ B         Abbiamo il nostro poligono ABCD in cui
  |                 |           intendiamo applicarci una texture.
  |                 |           Vogliamo che al punto più in alto a
```

```
      |                |          sinistra della texture corrisponda il
      |                |          vertice A, al punto in alto a destra
      |                |          il vertice B, a quello in basso a
      |                |          destra C, mentre al punto in basso a
      |                |          sinistra il vertice D.
      |                |          Facciamo finta che le coordinate del
   D +----------------+ C        poligono siano già  state ruotate.
                                  Al punto A corrisponde il punto (0,0)
della texture, al punto B le coordinate (255,0), a C corrisponde (255,255)
ed infine a D corrisponde il pixel (0,255) della texture.
```

Quindi svolgiamo 2 ulteriori scan conversion della faccia (oltre a quella classica per "fillare" il poligono) utilizzando una volta le $< u >$ iniziali e finali al posto delle x, mentre nella seconda volta si devono sfruttare le $< v >$ iniziali e finali al posto delle x; in questo modo (interpolando la $< u >$ e la $< v >$ per ogni scanline) sapremo le coordinate (u, v) della texture di ogni pixel della faccia.

31.18 Ottimizzazione del fill

In questo paragrafo studieremo un alternativo algoritmo per realizzare il fill dei poligoni nel caso avessimo bisogno di interpolare una o più variabili lungo tutto il poligono (come nel caso del texture mapping, del gouraud e del phong shading). L'unico vero svantaggio nell'utilizzare tale metodo consiste nel fatto che tutti poligoni dovranno per forza essere dei triangoli.

Consideriamo di dover mappare una texture (o parte di essa) su di un poligono. Applicando i metodi precedentemente descritti avremmo eseguito 2 divisioni per scanline, ovvero:

```
du = u2 - u1        dove (u1,v1) rappresentano le coordinate
dv = v2 - v1        della texture per l'edge a sinistra, mentre
stepu = du / dx     (u2,v2) valgono per l'edge a destra.
stepv = dv / dx
```

L'ottimizzazione notevole che si ottiene con l'algoritmo che vedremo, sta nel calcolare $stepu$ e $stepv$, detti constant slope, una sola volta per tutto il poligono. In altre parole al posto di svolgere 2 divisioni per scanline, eseguiremo 2 divisioni per poligono, non male!

In un triangolo qualsiasi, gli $stepu$ e $stepv$ sono sempre costanti lungo tutto il poligono, quindi potremo calcolarci i costant slope una volta sola (per faccia) e riutilizzarli per tutte le scanline di tale poligono. Per ottenere risultati il più precisi possibile, calcoleremo gli stepu e stepv relativi alla scanline più lunga di tutto il poligono, poi utilizzeremo tali constant slope per tutte le scanline. Questo è il concetto fondamentale per poter ottimizzare il fill.

Consideriamo un generico triangolo:

```
              A(x1,y1) .              A(x1,y1) è  il vertice superiore,
                     _/ \             che sta più in alto;
                   _/    \           B(x2,y2) è  il vertice che sta a
                 _/       \ L1       media altezza;
            L3 _/          \         C(x3,y3) è  il vertice inferiore,
             _/             \        che sta più in basso;
           _/            ____\.      L1 è  il lato AB; L2 è  il lato BC
         _/      ____----   B(x2,y2)  mentre L3 è  il lato CA.
C(x3,y3) ./____----  L2
```

La scanline più lunga è sempre quella che giace sul vertice B, ossia quel vertice vediamo come calcorarne la relativa lunghezza assieme ai constant slope:

```
 temp = (y2-y1) / (y3-y1)
 scan =  temp * (x3-x1) + (x1-x2)          <- lunghezza scanline maggiore
stepu = (temp * (u3-u1) + (u1-u2)) / scan  <- constant slope u
stepv = (temp * (v3-v1) + (v1-v2)) / scan  <- constant slope v
```

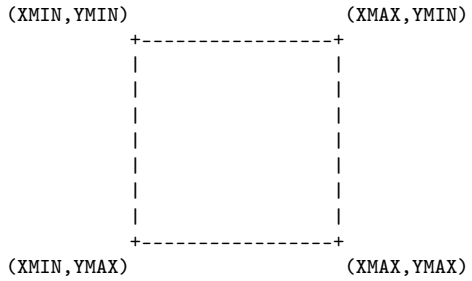
Il valore di $< scan >$ può essere positivo o negativo, il che ci permette di determinare se il vertice B è il punto estremo a destra o a sinistra dell'intero poligono:

```
se (scan = 0) allora il poligono è  vuoto, non tracciare il poligono;
se (scan > 0) allora il vertice B è  posto a sinistra, quindi i lati L1
                     e L2 appartengono all'edge di destra, mentre solo
                     L3 appartiene all'edge di sinistra;
se (scan < 0) allora il vertice B è  posto a destra, quindi i lati L1 e
                     L2 appartengono all'edge di sinistra, mentre solo
                     L3 appartiene all'edge di destra;
```

31.19 Clipping 2D

Nella visualizzazione di un oggetto o un mondo 3D, bisogna considerare il problema del tracciamento di poligoni o linee parzialmente visibili sullo schermo, ovvero di facce o linee che "escono" (anche parzialmente) fuori dal monitor. Il procedimento atto a risolvere questo problema è chiamato clipping 2D. Questo tipo di clipping è 2D in quanto ci si propone di "tagliare" i nostri oggetti solo rispetto al singolo piano coincidente lo schermo.

Come condizioni iniziali consideriamo i valori $XMIN$ e $XMAX$ come le componenti x estreme a sinistra e a destra dello schermo, mentre $YMIN$ e $YMAX$ come le componenti y estreme in alto e in basso dello schermo. Quindi i 4 vertici immaginari dello schermo avranno le seguenti coordinate:

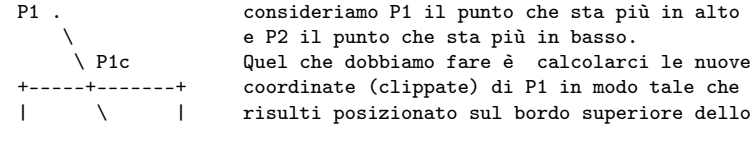


Ad esempio, nell'ipotetico caso di uno schermo di risoluzione 320x200:

```
XMIN = 0  ;  XMAX = 319  ;  YMIN = 0  ;  YMAX = 199
```

Analizziamo subito come clippare una singola linea, in seguito vedremo il clipping relativo ai poligoni (il quale si basa sullo stesso principio per tagliare una linea).

Caso 1 la linea esce dal bordo superiore dello schermo:




```
consideriamo P1 il punto che sta più in alto
e P2 il punto che sta più in basso.
Quel che dobbiamo fare è  calcolarci le nuove
coordinate (clippate) di P1 in modo tale che
risulti posizionato sul bordo superiore dello
```

```
|       \.    |     schermo, chiamiamo questo punto P1c.
|         P2  |
|             |     P1c(xclip,yclip)
|             |     P1(x1,y1)  ;  P2(x2,y2)
+-------------+
                    xclip = x1 + (YMIN-y1)*(x1-x2)/(y1-y2)
                    yclip = YMIN
```

Una volta ricavate le coordinate di $P1c$ non dovremo far altro che tracciare la linea tra $P1c$ e $P2$.

Caso 2 la linea esce dal bordo inferiore dello schermo:

```
+-------------+     P1 è  il punto superiore mentre P2 è  il punto
|             |     inferiore. P2c è  il punto di cui dobbiamo
|         .P1 |     calcolarci le coordinate.
|        /    |
|       /     |     P2c(xclip,yclip)
|  P2c /      |     P1(x1,y1)  ;  P2(x2,y2)
+-----+-------+
 P2 ./              xclip = x2 + (YMAX-y2)*(x2-x1)/(y2-y1)
                    yclip = YMAX
```

La linea clippata sarà compresa tra i punti $P1$ e $P2c$.

Caso 3 la linea esce dal bordo sinistro dello schermo:

```
      +-------------+     P1 è  il estremo della linea a sinistra e P2
P1.___|P1c          |     è  il punto estremo a destra della linea.
      |---___. P2   |
      |             |     P1c(xclip,yclip)
      |             |     P1(x1,y1)  ;  P2(x2,y2)
      |             |
      +-------------+     xclip = XMIN
                          yclip = y1 + (XMIN-x1)*(y1-y2)/(x1-x2)
```

La linea clippata sarà compresa tra i punti $P1c$ e $P2$.

Caso 4 la linea esce dal bordo destro dello schermo:

```
+-------------+     P1 è  il estremo della linea a sinistra e P2
|             |     è  il punto estremo a destra della linea.
|          ___|_.P2
| P1.___---P2c|     P2c(xclip,yclip)
|             |     P1(x1,y1)  ;  P2(x2,y2)
|             |
+-------------+     xclip = XMAX
                    yclip = y2 + (XMAX-x2)*(y2-y1)/(x2-x1)
```

La linea clippata sarà compresa tra i punti $P1$ e $P2c$.

Dato che l'argomento relativo al clipping 2D dei poligoni è stato già affrontato esaurientemente da CDS/DeathStar, ho preferito evitare di rispiegare tale argomento e ho deciso di riportare qui di seguito il testo che ha scritto CDS.

L'algoritmo di Sutherland-Hodgman permette di eseguire il clipping di un poligono convesso o concavo rispetto ad una finestra di clipping convessa. Solitamente la finestra di clipping è un rettangolo (lo schermo video), ma in alcuni casi è necessario "clippare" un poligono rispetto ad un altro (ad esempio nel calcolo delle ombre portate). Nel caso di finestra rettangolare SH ha

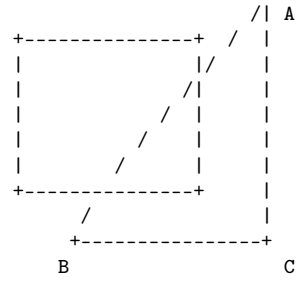
bisogno di 4 passi, uno per ogni lato della finestra di clipping. SH lavora su una lista di vertici e produce in output, ad ogni passo, una nuova lista di vertici che si trovano dalla parte visibile rispetto al lato di clipping. Siano $P1$ il vertice attuale nella lista, $P2$ il vertice seguente e T l'eventuale punto di intersezione fra la retta passante per $P1$ e $P2$ e il lato di clipping.

Bisogna considerare 4 casi:

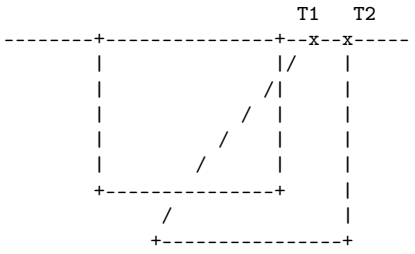
1. $P1$ interno $P2$ interno Nella lista di destinazione si inserisce $P2$.
2. $P1$ interno $P2$ esterno Nella lista di destinazione si inserisce T.
3. $P1$ esterno $P2$ interno Nella lista di destinazione si inseriscono T e $P2$.
4. $P1$ esterno $P2$ esterno Non si inserisce nessun vertice.

Le condizioni interno/esterno nel caso di clipping con una finestra rettangolare sono dei semplici confronti, nel caso di finestra convessa non rettangolare è necessario eseguire un prodotto scalare. Ad esempio il test $P1$ interno nel caso di clipping con il lato sinistro dello schermo è semplicemente $P1.x > 0$.

Un esempio di clipping di un triangolo con una finestra rettangolare:

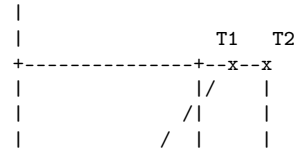


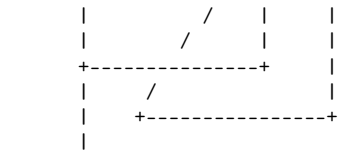
Passo 1 (clipping con il lato superiore) :



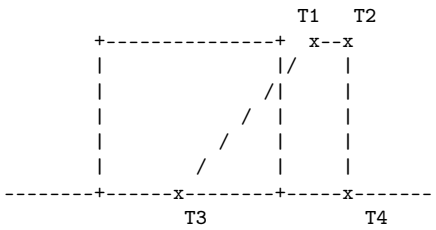

```
A è  esterno e B è  interno, si inseriscono T1 e B.
B è  interno e C è  interno, si inserisce C.
C è  interno e A è  esterno, si inserisce T2.
```

Si ottiene un nuovo poligono con i due vertici temporanei $T1$ e $T2$. Passo 2 (clipping con il lato sinistro):

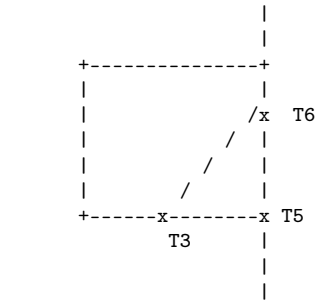




Passo 3 (clipping con il lato inferiore) :



Si ottiene un nuovo poligono con i due vertici temporanei $T3$ e $T4$. Passo 4 (clipping con il lato destro) :




```
Il poligono definitivo ha per vertici T3, T5 e T6.
```

CDS/DeathStar

31.20 Z-Buffer

Lo Z-Buffer è un'area di memoria dedicata all'eliminazione delle facce completamente o parzialmente nascoste. E' particolarmente utile per tracciare poligoni che si intersecano e non comporta limitazioni nella scena da tracciare (al contrario dell'algoritmo del pittore).

Lo spazio occupato in memoria dallo Z-Buffer coincide con le dimensioni dello schermo (es.: per uno schermo 320*200 avremo bisogno di un buffer di 64000 valori). Consideriamo lo Z-Buffer come il nostro schermo chunky, con la differenza che ogni posizione non rappresenta un pixel, bensì la componente Z di quel pixel; in questo modo possiamo sapere la coordinata Z di tutti i punti visualizzati su schermo.

Ora si procede nella seguente maniera. Per una ed una sola volta per ogni frame andiamo ad inizializzare lo Z-Buffer copiandoci su ogni posizione un valore arbitrario piuttosto elevato.

In seguito ci andiamo ad interpolare la coordinata Z di ogni vertice lungo tutto il poligono (ossia facciamo come nel gouraud, ma consideriamo Z al posto di Nz).

Nel loop nel quale tracciamo i pixel (ovvero il loop più interno del fill), confrontiamo la Z che stiamo interpolando con il valore corrispondente nello Z-Buffer (la cui posizione deve coincidere con quella dello schermo chunky) e, soltanto nel caso la nostra Z sia minore rispetto a quella

dello Z-Buffer, copiamo la nostra Z nello Z-Buffer e scriviamo il pixel nello schermo chunky. Nel caso in cui la nostra Z sia maggiore o uguale rispetto al valore dello Z-Buffer, allora non dobbiamo nè aggiornare lo Z-Buffer, nè tracciare il pixel su schermo (comunque in ogni caso dobbiamo continuare ad interpolare sia la Z che altre eventuali variabili).

31.21 Bump mapping 2D

Nel texture mapping abbiamo la possibilità di rivestire un oggetto con un'immagine qualunque, quindi la superficie dell'oggetto sarà data dalla texture ad essa applicata, quindi la superficie può considerarsi liscia, ovvero piana, completamente piatta. Il bump mapping ha lo scopo di rendere una texture "pseudo-tridimensionale", il che significa che l'immagine è solida (e non piatta) a livello ottico, ma in realtà viene elaborata come uno strumento 2D (in pratica non c'è nessun asse z per la texture). Per creare l'effetto "simil-3D" applichiamo sull'immagine ombre e luci, utilizzando casomai una sorgente di luce in movimento.

Per realizzare il bump mapping dovremo utilizzare una texture un pò particolare, la quale deve avere una determinata disposizione dei colori. Mettiamo caso che la nostra immagine utilizzi 256 colori, partendo dal primo colore sino al duecentocinquantaseiesimo specifichiamo le relative componenti RGB; nel nostro caso il primo colore dovrà essere il più scuro, il successivo dovrà essere quello un pò meno scuro, il terzo quello poco più chiaro del secondo e così via, in modo tale che l'ultimo colore sia il più chiaro di tutti. La texture dovrà trovarsi in memoria secondo una successione continua e ordinata di righe (ovvero: ogni riga di pixel dovrà trovarsi esattamente nella locazione di memoria successiva rispetto alla locazione in cui è presente l'ultimo pixel della riga precedente). Inoltre ogni pixel dovrà essere rappresentato come un singolo byte.

Premesso ciò, possiamo fare alcune osservazioni. Facciamo finta che la texture da "bumpare" sia la rappresentazione di una catena montuosa vista da un aereo, la cui tavolozza di colori è composta da 256 tonalità di grigio. Prendiamo un pixel qualunque di questa immagine, il pixel chunky è un byte, quindi un valore compreso tra 0 e 255; tanto questo valore sarà maggiore e tanto quel pixel risulterà chiaro (come colore). Un pixel più è chiaro e più rifletterà luce; i punti che riflettono più luce sono quelli maggiormente vicini alla sorgente di luce: in sostanza possiamo affermare che più è alto il valore del pixel chunky e più questo risulterà vicino alla sorgente di luce; al contrario possiamo dire che più è basso il valore del pixel e più sarà lontano dalla sorgente di luce.

Adesso vediamo come realizzare praticamente il bump mapping. L'algoritmo si può suddividere principalmente in 2 parti: nella prima svolgiamo un precalcolo, ovvero calcoliamo una tabella in memoria; la seconda parte consiste nel calcolo vero e proprio della texture "bumpata", la quale parte (a differenza della prima) viene svolta in tempo reale. Per una singola texture, il precalcolo della tabella viene svolto solamente una volta.

La tabella occupa esattamente 4 volte lo spazio occupato dalla texture. Per ricavarla dobbiamo fare l'intera scansione della nostra immagine. Supponiamo che $P1$ sia l'attuale pixel chunky scansito, che $P2$ sia il successivo punto sulla destra di $P1$ e che $P3$ sia esattamente quello posto al di sotto di $P1$, quindi le coordinate di questi 3 punti saranno:

```
P1( x   , y   )    <- punto attuale
P2( x+1 , y   )    <- punto sulla destra di P1
P3( x   , y+1 )    <- punto al di sotto di P1
```

Ecco i calcoli da eseguire:

```
px = P1-P2         <- differenza di pixel chunky
py = P1-P3         <- differenza di pixel chunky
```

```
ox = h*sin(px)
oy = h*sin(py)

of = oy*tx+ox      <- valore della tabella da precalcolare

tx = 256           <- larghezza della texture in pixel
h                  <- valore arbitrario tra 0 e 255
```

Da notare che px e py coprono un range di valori tra -255 e $+255$. Nel nostro caso -255 rappresenta $-pigreco/2$ (ovvero -90 gradi), mentre $+255$ indica $pigreco/2$ (+90 gradi). Se avessimo una funzione seno che accettasse in ingresso un angolo in radianti, $\sin(px)$ sarebbe equivalente a: $\sin(px * 512/PI)$. È consigliabile calcolare una tabella di $\sin(x)$ in cui x varia da -90 gradi a +90 gradi formata da 512 valori.

La tabella è composta semplicemente da un insieme di of, e precisamente uno per ogni pixel dell'immagine. of è un valore a 32 bit, quindi per una texture 256*256 la tabella risulta lunga 256kb. Il valore $< h >$ è il coefficiente di perturbazione, cioè indica il grado dell'effetto bump. In altre parole tanto h sarà maggiore e tanto si noterà che la texture è tridimensionale, tanto h sarà minore e tanto la texture risulterà piatta. E qui finisce la fase di precalcolo.

Vediamo la parte da svolgere in tempo reale. Innanzitutto ci serve una ulteriore texture, la quale verrà utilizzata per "simulare" la luce applicata alla texture da bumpare. Questa immagine è la riproduzione vera e propria della luce; in pratica rappresenta una serie di cerchi concentrici, di cui il cerchio più interno (quindi il più piccolo) è riempito con il massimo valore attribuibile ad un pixel chunky (ovvero 255), mentre il cerchio più esterno (il più grande) è riempito con il minor valore attribuibile al pixel chunky (che sarebbe 0). E' possibile utilizzare la stessa texture che si applica nel reflection mapping per simulare una sorgente di luce. Un buon esempio di questo tipo di immagini è una sfera (di dimensioni arbitrarie) più luminosa al centro e resa scura ai bordi.

Consideriamo di utilizzare 3 variabili (o registri) che puntano rispettivamente al buffer dove salvare la bump-texture (che può anche essere lo schermo chunky), alla texture che rappresenta la luce e alla tabella precalcolata. Per ogni byte del buffer della texture da calcolare noi preleviamo sequenzialmente un valore dalla tabella precalcolata (ossia un valore della tabella per ogni byte del buffer). In seguito utilizziamo questo valore come offset da applicare al puntatore della texture che rappresenta la luce (ovvero addizioniamo al valore della tabella il puntatore della texture coincidente la sorgente di luce) e copiamo il relativo byte nel buffer destinazione. Dopo aver svolto il tutto per ogni pixel avremo ottenuto il bump mapping.

31.22 Notazione in virgola fissa

Nell'elaborazione di oggetti 3D si ha spesso a che fare con numeri reali, non interi. La maggior parte dei linguaggi evoluti permettono la diretta manipolazione di tali numeri, casomai sfruttando un eventuale coprocessore matematico oppure emulandoli via software. L'emulazione che apportano gli attuali compilatori risulta decisamente lenta per applicazioni in tempo reale, inoltre se si lavora in Assembly non si ha a disposizione la diretta gestione di numeri reali a meno che non venga utilizzato il coprocessore matematico. In alternativa alla FPU (che sfrutta numeri in virgola mobile) si può optare per il formato in virgola fissa che, nonostante una minore precisione rispetto al formato in virgola mobile, rimane la scelta migliore in quanto le operazioni svolte in tale formato risultano più veloci. In linea di principio, in un elaboratore elettronico, tutti i numeri (anche quelli reali) vengono rappresentati come interi, la notazione in virgola fissa si basa sulla diretta semplificazione di tale rappresentazione, vediamo come.

Un numero reale viene rappresentato come quel valore intero dato dal prodotto del numero reale moltiplicato per una costante definita a priori. E' proprio da questa costante che dipende la precisione con la quale possono essere rappresentati numeri non interi. Ecco un esempio:

```
3.25             <- numero reale
256              <- costante

3.25*256 = 832   <- 3.25 in virgola fissa
```

In questo modo possiamo rappresentare tutti i numeri reali con un discreto margine di errore che per le nostre applicazioni risulta praticamente ininfluente. È conveniente utilizzare come costante una potenza di 2 (es.: 256, 65536), con la quale è possibile velocizzare la manipolazione di numeri in tale notazione. Difatti è noto che il computer rappresenta un qualsiasi numero come una sequenza di bit, quindi, utilizzando come costante una potenza di 2, è possibile definire 2 campi di bit per ogni cifra: una dedicata alla parte intera mentre l'altra dedicata alla parte frazionaria. Se un numero in virgola fissa ha un numero di bit dedicati alla parte intera uguale ad $< a >$ e un numero di bit dedicati alla parte frazionaria uguale a $< b >$, allora si dice che quel numero è nel formato $a : b$. Bisogna inoltre specificare che la parte intera di una cifra in virgola fissa appartiene al campo di bit più alto, mentre la parte frazionaria appartiene al campo di bit più basso.

```
3.25        <- numero reale
256=2^8     <- costante
832         <- 3.25 in virgola fissa (ovvero 3.25*256)
8:8         <- formato del numero in virgola fissa. Se
               utilizziamo 1 word (16 bit) abbiamo gli 8 bit
               più significativi dedicati alla parte intera e
               gli altri 8 bit meno significativi per quella
               frazionaria.
```

Vediamo come convertire un numero intero nel formato in virgola fissa e viceversa:

```
numero intero        = (numero virgola fissa) / (costante)
numero virgola fissa = (numero intero)        * (costante)
```

Infine occupiamoci di comprendere come effettuare le 4 operazioni con tali numeri:

```
(a:b) + (c:d) = impossibile!!
(a:b) + (a:b) = a:b
(a:b) * (c:d) = (a+c):(b+d)
(a:b) / (c:d) = (a-c):(b-d)
```

Ci accorgiamo subito che risulta impossibile sommare 2 numeri in virgola fissa di diverso formato, bisogna prima rendere omogenee le 2 cifre (significa che le 2 cifre devono avere lo stesso formato). Da notare che una qualsiasi cifra intera può essere intesa come un numero in virgola fissa nel formato "a:0"; quindi è possibile svolgere direttamente moltiplicazioni e divisioni tra numeri in virgola fissa ed interi.

31.23 Coordinate polari

Come già sappiamo, per rappresentare un generico punto su di un piano possiamo utilizzare gli assi cartesiani. Le componenti x ed y non rappresentano altro che le proiezioni del nostro punto sull'ascissa e sull'ordinata. Immaginiamo invece di voler indicare un punto utilizzando un altro sistema di riferimento, nel nostro caso le coordinate polari.

```
  ^                    Consideriamo r la distanza tra il punto P e
y |     .P(x,y)        l'origine, mentre t come l'angolo compreso tra il
  |    /               segmento OP ed il semiasse positivo x.
  |   /                E' possibile indicare qualsiasi punto utilizzando
  | r/                 queste 2 variabili (r e t), le quali rappresentano
  | /                  proprio le coordinate polari.
  |/) t                Per ogni P(x,y) corrisponde un P'(r,t). Vediamo
  +------------>       come effettuare queste conversioni.
 0            x
                       Sia R(x,0) la proiezione di P(x,y) sull'asse x
  ^                    (ovvero il punto di ordinata 0 e di equivalente
y |     .P(x,y)        ascissa di P). Il triangolo ORP è  rettangolo in R,
  |    /|              quindi per il teorema di Pitagora:
  |   / |
  | r/  |                        r = OP = sqrt( x*x + y*y )
  | /   |
  |/) t |              Possiamo anche affermare che:
  +-----+------>
 0      R     x        PR = r*sin(t)   =>   sin(t) = PR/r
                       OR = r*cos(t)   =>   cos(t) = OR/r
```

Se facciamo attenzione possiamo affermare che (considerando il punto $P(x, y)$) $PR = y$ e $OR = x$. La tangente di un angolo è equivalente al rapporto del seno di quell'angolo col relativo coseno, quindi:

```
tan(t) = sin(t)/cos(t) = (PR/r)/(OR/r) = PR/OR = x/y
x/y = tan(t)

t = arctan(x/y)
r = sqrt(x*x+y*y)

x = r*cos(t)
y = r*sin(t)
```

Ora sappiamo come convertire le coordinate cartesiane in polari e viceversa, ciò risulterà utile per comprendere come effettuare la rotazione di un punto.

31.24 Gestione degli oggetti

Vogliamo tracciare il nostro oggetto su schermo sia esso in wireframe, gouraud shading, texture mapping o in altra tecnica di rendering che più ci va a genio; sappiamo perfettamente come visualizzare una singola faccia, ma come potremmo gestire tutte le facce che compongono il solido tridimensionale? Bisogna definire un "formato" con cui l'oggetto risiede in memoria, in base al quale potremo visualizzare qualsiasi figura 3D utilizzando sempre le stesse routine che compongono il nostro motore 3D. Iniziamo col vedere un esempio pratico:

```
       V5 _____________ V6     teniamo conto di dover definire come
        /|             /|      oggetto un semplice cubo, per prima
       / |            / |      cosa potremmo indicarne il numero di
      /  |           /  |      vertici e di facce di cui è  composto;
     /   |          /   |      in seguito elenchiamo tutte le
  V1 /____|_______ /V2  |      coordinate (x,y,z) dei vertici del cubo;
    |     |       |     |      infine indichiamo le caratteristiche di
    |   V8|_______|_____|V7    tutte le facce. Nel caso più semplice
    |    /        |    /       per definire una faccia basta indicarne
    |   /         |   /        i vertici (casomai ordinati in senso
    |  /          |  /         orario per facilitare la rimozione
    | /           | /          dell'hidden face e il calcolo della
```

```
   |/____________|/           normale). Ecco come è  possibile
V4               V3           definire in memoria un cubo:

        8            <- numero di vertici dell'oggetto
        6            <- numero di facce dell'oggetto
        -50,-50,-50  <- coordinate x,y,z del vertice V1
        +50,-50,-50  <- coordinate x,y,z del vertice V2
        +50,+50,-50  <- coordinate x,y,z del vertice V3
        -50,+50,-50  <- coordinate x,y,z del vertice V4
        -50,-50,+50  <- coordinate x,y,z del vertice V5
        +50,-50,+50  <- coordinate x,y,z del vertice V6
        +50,+50,+50  <- coordinate x,y,z del vertice V7
        -50,+50,+50  <- coordinate x,y,z del vertice V8
        1,2,3,4      <- indici al buffer dei vertici della faccia 1
        2,6,7,3      <- indici al buffer dei vertici della faccia 2
        6,5,8,7      <- indici al buffer dei vertici della faccia 3
        5,1,4,8      <- indici al buffer dei vertici della faccia 4
        5,6,2,1      <- indici al buffer dei vertici della faccia 5
        4,3,7,8      <- indici al buffer dei vertici della faccia 6
```

Analizziamo come vengono definiti i poligoni, prendiamo la faccia 1:

```
        1,2,3,4      <- significa che la faccia è  composta dai primi 4
                        punti posti nella lista dei vertici, ovvero:

        -50,-50,-50  <- coordinate x,y,z del vertice V1
        +50,-50,-50  <- coordinate x,y,z del vertice V2
        +50,+50,-50  <- coordinate x,y,z del vertice V3
        -50,+50,-50  <- coordinate x,y,z del vertice V4

        2,6,7,3      <- significa che la faccia è  composta dai lati
                        delimitati dai punti 2 e 6, 6 e 7, 7 e 3 e dai
                        punti 3 e 2.
```

Ogni lato è rappresentato dalla congiunzione lineare di 2 vertici, nell'esempio appena proposto i lati della faccia 1 sono i segmenti delimitati dal primo al secondo punto $(V1eV2)$, dal secondo al terzo $(V2eV3)$, dal terzo al quarto $(V3eV4)$ e dal quarto al primo punto $(V4eV1)$. Nel nostro caso ogni faccia è un quadrilatero, naturalmente è possibile realizzare il proprio motore 3D che utilizzi un diverso numero di lati, l'importante è che ogni poligono sia convesso, altrimenti l'algoritmo di scan conversion risulterebbe decisamente più complesso di quello studiato nel paragrafo relativo al fill e scan line.

Parte IV

Appendici

APPENDICE A

MATEMATICA

Autore: Antonello Mincone

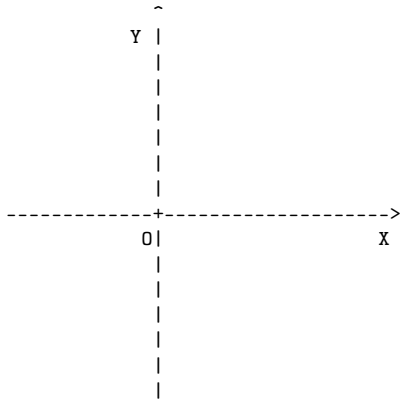
Per tutti quelli che aspirano a fare giochi tipo DOOM, ELITE, o comunque un qualsiasi gioco che richieda l'uso di poligoni, o addirittura di quella che è uno degli effetti ormai diventati uno standard nei DEMOS, cioè il texture-mapping, credo sia essenziale sapere alcune di quelle che sono le formule basilari di matematica,in particolare quelle relative alla geometria analitica e alla trigonometria.

Se a scuola non avete mai digerito questi argomenti,o semplicemente non li avete mai affrontati e ne avete sempre sentito parlare come qualcosa di incredibilmente complicato, posso assicurarvi che non è assolutamente vero.La vera difficoltà sta nel seguire attentamente l'argomento, ma soprattutto nel capirlo.

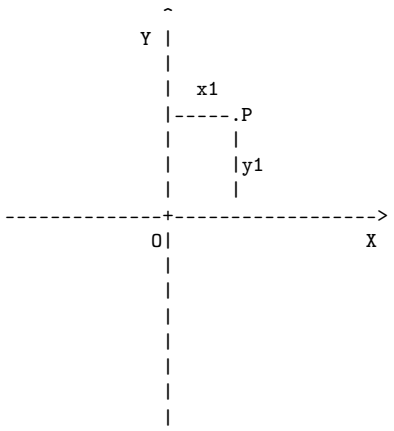
In pratica il consiglio che vi do è di fare (anche in base alle conoscenze che avete) poco alla volta sia con questa lezione che con il corso in generale, dato che in realtà potete possedere tutti i libri di informatica e matematica di questo mondo, ma questi costituiscono solo la base essenziale (ma non sufficiente) per scrivere un programma valido:si diventa bravi soprattutto con l'esperienza, provando e riprovando le routine, modificandole, facendo insomma degli esperimenti.

In base a ciò ho deciso di scrivere quest'articolo non insegnandovi una parti colare tecnica di 3D (anche perché per quella c'è una lezione apposta), ma le basi che vi permetteranno di ricavarvi da soli le formule anche solo per farvi la tabella pre-calcolata adatta alle vostre esigenze. Partirò quindi da zero (beh non proprio, dato che spero conosciate le quattro operazioni fondamentali,anche perché se no, non dovete fare il corso ma le elementari), non si stupisca quindi tutta quella gente che conosce già l'argomento e magari avrebbe qualcosa da obbiettarmi sullo stile di esposizione.

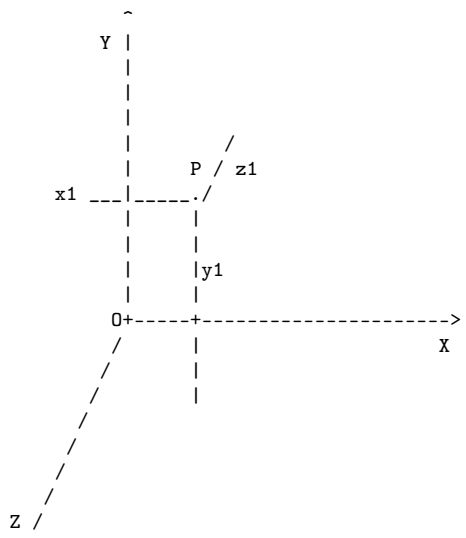
Inanzi tutto parliamo di quello che viene chiamato dagli esperti in materia *sistema di riferimento cartesiano ortogonale*. In realtà sotto questo nome si nasconde una cosa semplicissima: due comuni rette (si dice RETTA una linea di cui non c'è ne un inizio e ne una fine, si dice SEMI-RETTA una linea che ha un inizio ma non una fine, si dice SEGMENTO una linea che ha un inizio e una fine) disposte in modo che, incrociate tra loro, formino quattro angoli di 90 gradi, che hanno convenzionalmente i nomi di X e Y.In pratica disposte cosi:



Il punto O corrisponde all'incrocio degli assi ed è chiamato ORIGINE. Gli assi X e Y sono chiamati rispettivamente asse delle ascisse e asse delle ordinate, e servono a darci dei riferimenti ad ogni punto, che si ottengono tracciando la parallela all'asse X e quella all'asse Y, e che in pratica ci dicono l'altezza del punto rispetto ad X e la sua distanza da Y, che esprimeremo scegliendo un'unità di misura e vedendo quante volte questa unità entra nella misura considerata: quando cioè vogliamo indicare dove si trova un determinato punto dovremmo definire la sua distanza con l'asse X e con l'asse Y (più precisamente la sua ordinata e la sua ascissa).Immaginiamo ad esempio di avere un punto P :



Il segmento contrassegnato con x1 indica la sua ascissa mentre quello contrassegnato con y1 la sua ordinata.Tralasceremo l'unità di misura dato che con l'AMIGA userete il pixel. Badate bene che quando il punto si trova a destra dell'asse Y l'ascissa è positiva (è cioè maggiore di 0), mentre quando si trova a sinistra è negativa (è cioè minore di 0); caso limite quando si trova sull'asse Y,dove sarà uguale a 0.Allo stesso modo per l'asse X: quando il punto si trova sopra di questo l'ordinata è positiva, sotto è negativa, sull'asse x è 0. Se però rapportiamo il tutto alla realtà,ci accorgiamo che due dimensioni non sono sufficienti, dato che tutti gli oggetti, hanno, oltre ad una larghezza e una larghezza, anche una profondità. Abbiamo quindi bisogno di una terza dimensione, appunto la profondità, che ci permette di rapportare anche il singolo punto allo spazio.Un grafico quindi che ci voglia dare un completo quadro di un oggetto nello spazio sarà di questo genere:



Il nuovo asse, Z, indica la nuova dimensione.è da tener presente che, nello spazio, l'angolo situato tra l'asse X e l'asse Z e quello tra l'asse Y e l'asse Z, è retto, cioè di 90 gradi, che purtroppo si deformano in proiezione assonometrica (che è appunto quella con cui sono rappresentati tutti i grafi- ci).

Finora abbiamo parlato solo di punti, mentre lo spazio che ci circonda è fatto da oggetti ben più complessi, che in genere sono fatti da linee che uniscono gli spigoli che li determinano.Molte volte è necessario dover rappresentare oggetti curvilinei, quali il semplice cerchio o curve più complesse, determinate da complicate formule goniometriche: in questi casi con l'AMIGA consiglio sempre di ridurre la curva a un poligono, magari anche con 20 lati, ma che risulta certamente più veloce da disegnare e da calcolare in rotazioni o traslazioni (movimenti che comportano solo lo spostamento),comunque troverete alla fine della lezione le formule delle più importanti curve.

Per unire i punti dei poligoni generalmente si può ricorrere alla funzione LINE del blitter, ma non sempre questa è la più veloce, e potrebbe essere necessario affidare questo lavoro al processore: è quindi utile conoscere le formule principali legate al tracciamento delle rette. Iniziamo col dire che ogni retta è individuata sugli assi cartesiani dalla formula:

```
Y = m*X + q
```

La formula in questione ci da le ordinate di tutti i punti della retta a seconda della sua ascissa, basta insomma sostituire la x con un qualsiasi valore per ottenere la corrispondente y della retta.I termini m e q che compaiono nella formula sono delle costanti: la prima m è detto coefficiente angolare e determina l'angolo che la retta forma con l'asse X (più precisamente è la tangente di quell'angolo, ma affronteremo questo argomento più avanti), più grande sarà m e più grande sarà l'angolo formato; q determina invece il punto dove la retta incrocia l'asse y, in sostanza il punto della retta che ha coordinate: (0,q), da ciò è facile capire che se q=0, allora la retta passa per l'origine degli assi.

Esiste poi una formula che a mio parere è importantissima per fare un programma 3D. Dato un punto P1 di coordinate (P1x,P1y), e un punto P2 di coordinate (P2x,P2y), possiamo ricavare la retta passante per questi due punti con la formula:

```
Y-P1y = (P2y-P1y)/(P2x-P1x)*(X-P1x)
```

Dalla formula in questione ricaviamo che:

```
Y = (P2y-P1y)/(P2x-P1x)*X + (-P1x*(P2y-P1y)/(P2x-P1x))+P1y
```

Questa è appunto la formula della retta passante per i punti presi in considerazione.Il termine che compare prima della X corrisponde alla m, mentre tutta la formulaccia che compare dopo sarebbe la q, ma naturalmente questo calcolo sarà fatto una sola volta per ogni retta. Applicazioni della stessa formula possono servire ad esempio, per una linea che esce dallo schermo: avendo gli estremi di questa possiamo trovare la formula che la determina e quindi, sostituendo alla X la ascissa dei bordi dello schermo, possiamo trovare le coordinate dei punti estremi del segmento visibile.

Altre formule che spesso si rivelano utili conoscendo le coordinate di due punti P1 (P1x,P1y) e P2 (P2x,P2y) sono:

1. Quella per trovare la loro distanza (che è in pratica un' applicazione del teorema di Pitagora che troverete più avanti):

   ```
   distanza = sqr((P2x-P1x)^2+(P2y-P1y)^2)
   ```

   (sqr non è altro che l'istruzione usata dalla maggior parte dei linguaggi di alto livello per indicare la radice quadrata, mentre il simbolo ^significa elevato: in questo caso leggeta quindi la distanza è uguale alla radice quadrata della differenza delle ascisse elevata al quadrato sommata alla differenza delle ordinate elevata al quadrato, in simboli:

   ```
        ----------------------------------------
       /
      / (P2x-P1x)*(P2x-P1x)+(P2y-P1y)*(P2y-P1y)
   \  /
    \/
   ```

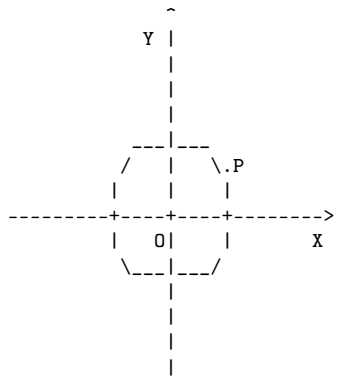
   Cercate di capire bene Sqr e ^perché in seguito li riuseremo spesso). Questa formula è utile ad esempio per trovare la lunghezza di un lato di un qualsiasi poligono conoscendo i due spigoli.

2. Conoscendo i soliti due punti P1(Px1,Py1) e P2(Px2,Py2) possiamo trovare le coordinate del punto medio M(XM,YM) con la formula:

   ```
   XM = (Px1 + Px2)/2

   YM = (Py1 + Py2)/2
   ```

A questo punto direi che potete anche fare la lezione di prospettiva, dato che ora siete capaci di rappresentare un qualunque oggetto nello spazio (basta infatti disegnarvi gli spigoli e unirli col Blitter in modo da formare una figura piana o un solido).Badate bene però che con queste conoscenze non potete ancora ruotare gli oggetti, ma solo zoomarli (per far questo basta aumentare o diminuire la Z di ogni punto).

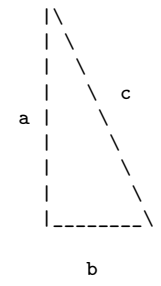
Per ruotare un punto dobbiamo entrare nella trigonometria introducendo il coseno e il seno.Questi due non sono altro che l'ascissa e l'ordinata di un punto che ha la caratteristica di trovarsi su una circonferenza che ha come centro l'origine.



Anche se quello che ho disegnato è un ottagono irregolare (ma che ci volete fare con i caratteri ASCII non sono riuscito a fare di meglio), con un pò di fantasia dovreste avere un'idea di quello che voglio dire.In breve il coseno e la distanza del punto P con l'asse Y, mentre il seno la distanza del punto P con l'asse X.Per convenzione (ma non solo per quello) il raggio del cerchio viene considerato pari ad 1. In questo modo sia il seno che il coseno oscilleranno sempre tra valori compresi tra 1 e -1 (in sostanza numeri decimali). Da notare inoltre che il punto P individua anche un angolo sulla circonferenza, l'angolo cioè formato tra l'asse X e la retta passante per il punto P e l'origine degli assi.

Se ad esempio diciamo che il seno di 30 gradi è 0.5, ciò significa che il punto P che, unito con O (origine degli assi), che forma con l'asse X un angolo di 30 gradi, dista dall'asse X 0.5. Per trovarci anche il coseno dell'angolo considerato possiamo fare una semplice osservazione, basandoci sul teorema di Pitagora (non il Coder). Per chi non conosce questo, che è uno dei principali teoremi di geometria ecco qui una veloce spiegazione:

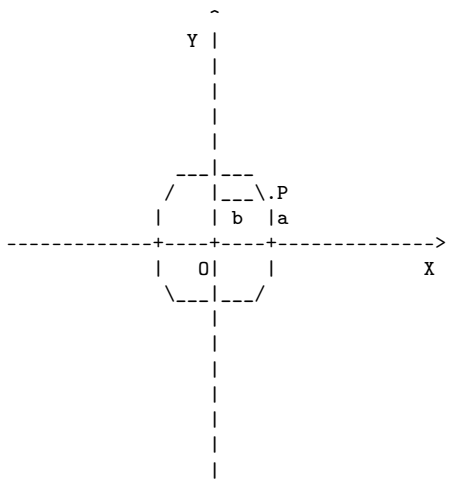
Definizione 1. *dato un triangolo rettangolo (che ha cioè un angolo di 90 gradi), sapendo la lunghezza dei due cateti (che sarebbero i lati più corti), possiamo trovare l'ipotenusa (il lato più lungo), sapendo che questa è uguale alla radice quadra ta della somma dei quadrati dei due cateti.*



In questo caso a e b sono i cateti, per trovare c dovremo calcolare la radice quadrata di $a * a + b * b$ (che possiamo anche scrivere $a^2 + b^2$). In generale quindi:

```
c^2 = a^2 + b^2
```

Tornando alla circonferenza che stavamo considerando, notiamo anche qui la presenza di un triangolo rettangolo che ha come cateti l'ascissa e l'ordinata del punto P, e come ipotenusa il segmento OP, che in pratica è uguale al raggio, e quindi ad 1. Nell'esempio di prima, dove conoscevamo il seno di 30 gradi, possiamo trovare il corrispondente coseno (che sarebbe in pratica l'ascissa):



In questo caso infatti a=0.5 ed OP (che non ho disegnato per motivi grafici) è uguale ad 1.Esseno quindi l'angolo tra a e b di 90 gradi, sostituendo i termini noti nell'equazione precedente abbiamo che :

```
1^2 = 0.5^2 + b^2
```

Sostituendo a 0.5 la forma frazionaria 1/2 possiamo scrivere:

```
1 = 1/2^2 + b^2
```

Da cui :

```
1 = 1/4 + b^2
```

Quindi:

```
b^2= 1 - 1/4

b^2= 3/4
```

Possiamo concludere che b = sqr(3)/2 (sqr non è altro che l'istruzione usata dalla maggior parte dei linguaggi di alto livello per indicare la radice quadrata, nel nostro caso leggete quindi b è uguale a radice di 3 fratto 2)

APPENDICE B

LICENZA PER DOCUMENTAZIONE LIBERA GNU

GNU Free Documentation License
Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be

distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material

this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy

a computer-network location from which the general network-using
public has access to download using public-standard network protocols
a complete Transparent copy of the Document, free of added material.
If you use the latter option, you must take reasonably prudent steps,
when you begin distribution of Opaque copies in quantity, to ensure
that this Transparent copy will remain thus accessible at the stated
location until at least one year after the last time you distribute an
Opaque copy (directly or through your agents or retailers) of that
edition to the public.

It is requested, but not required, that you contact the authors of the
Document well before redistributing any large number of copies, to
give them a chance to provide you with an updated version of the
Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under
the conditions of sections 2 and 3 above, provided that you release
the Modified Version under precisely this License, with the Modified
Version filling the role of the Document, thus licensing distribution
and modification of the Modified Version to whoever possesses a copy
of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct
   from that of the Document, and from those of previous versions
   (which should, if there were any, be listed in the History section
   of the Document). You may use the same title as a previous version
   if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities
   responsible for authorship of the modifications in the Modified
   Version, together with at least five of the principal authors of the
   Document (all of its principal authors, if it has fewer than five),
   unless they release you from this requirement.
C. State on the Title page the name of the publisher of the
   Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications
   adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice
   giving the public permission to use the Modified Version under the
   terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections
   and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add
   to it an item stating at least the title, year, new authors, and
   publisher of the Modified Version as given on the Title Page. If
   there is no section Entitled "History" in the Document, create one
   stating the title, year, authors, and publisher of the Document as
   given on its Title Page, then add an item describing the Modified
   Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for
   public access to a Transparent copy of the Document, and likewise
   the network locations given in the Document for previous versions
   it was based on. These may be placed in the "History" section.
   You may omit a network location for a work that was published at
   least four years before the Document itself, or if the original
   publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications",
   Preserve the Title of the section, and preserve in the section all

   the substance and tone of each of the contributor acknowledgements
   and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,
   unaltered in their text and in their titles. Section numbers
   or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section
   may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements"
   or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant. To do this, add their titles to the
list of Invariant Sections in the Modified Version's license notice.
These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a
passage of up to 25 words as a Back-Cover Text, to the end of the list
of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or
through arrangements made by) any one entity. If the Document already
includes a cover text for the same cover, previously added by you or
by arrangement made by the same entity you are acting on behalf of,
you may not add another; but you may replace the old one, on explicit
permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License
give permission to use their names for publicity for or to assert or
imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this
License, under the terms defined in section 4 above for modified
versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and
list them all as Invariant Sections of your combined work in its
license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and
multiple identical Invariant Sections may be replaced with a single
copy. If there are multiple Invariant Sections with the same name but
different contents, make the title of each such section unique by
adding at the end of it, in parentheses, the name of the original
author or publisher of that section if known, or else a unique number.
Make the same adjustment to the section titles in the list of
Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"
in the various original documents, forming one section Entitled
"History"; likewise combine any sections Entitled "Acknowledgements",
and any sections Entitled "Dedications". You must delete all sections
Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document

except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in

```
part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this
License, and if all works that were first published under this License
somewhere other than this MMC, and subsequently incorporated in whole or
in part into the MMC, (1) had no cover texts or invariant sections, and
(2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site
under CC-BY-SA on the same site at any time before August 1, 2009,
provided the MMC is eligible for relicensing.


ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of
the License in the document and put the following copyright and
license notices just after the title page:

    Copyright (c)  YEAR  YOUR NAME.
    Permission is granted to copy, distribute and/or modify this document
    under the terms of the GNU Free Documentation License, Version 1.3
    or any later version published by the Free Software Foundation;
    with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
    A copy of the license is included in the section entitled "GNU
    Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts,
replace the "with...Texts." line with this:

    with the Invariant Sections being LIST THEIR TITLES, with the
    Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other
combination of the three, merge those two alternatives to suit the
situation.

If your document contains nontrivial examples of program code, we
recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.
```