Questo mese ci occuperemo di una procedura alquanto piccola se paragonata al resto del sistema operativo di Amiga, ma non per questo priva di importanza: la gestione dei Boot Block. È proprio tramite essi che le realtà più pure di Amiga, e cioè il multitasking e l'AmigaDos vengono fatte partire. Ci occuperemo quindi di una procedura che qualunque utente di Amiga utilizza ogni giorno ma senza comprenderne purtroppo nella maggior parte dei casi la più pura essenza...

Amiga e i Boot Block

di Giuliano Peritore - Latina

Quasi sicuramente ci sarà fra di voi qualcuno che si sarà già chiesto come sia strutturato un Boot Block e certamente si sarà scontrato con una cronica mancanza di informazioni. Lo stesso Rom Kernel Manual dedica ai Boot Block una sola pagina che, se pure dice quasi tutto quello che c'è da dire, non brilla certo per chiarezza. Proprio per questo motivo, e anche a causa di parecchie domande che mi sono state rivolte a riguardo, ho deciso di realizzare questo articolo con cui spero di fare scomparire qualunque mistero legato a quei fatidici settori 0 e 1 che infestano tutti i nostri dischi. Parleremo quindi di qualcosa che qualunque utente di Amiga utilizza ogni giorno, e per di più parecchie volte e senza problemi, ma per lo più senza comprendere i precisi meccanismi che ne regolano il funzionamento. Il Boot Block è il primo impatto che si ha con un Amiga. O si possiede un disco Boot-abile oppure si è destinati a guardare in eterno (chi non ha il Kickstart 1.3) la manina che richiede il disco del Workbench.

Generalità

Innanzitutto occupiamoci del meccanismo di gestione dei Boot Block. Quando il sistema viene acceso avviene un reset di tutte le componenti hardware, 68000 incluso. In questo momento vi è un particolare bit di fondamentale importanza, si tratta del Memory Overlay Bit. La sua funzione consiste nel selezionare due diversi banchi di memoria fittizi presenti nell'Amiga. Il 68000 infatti, quando viene attivato mette nello SSP (System Stack Pointer) il valore contenuto in $0 e nel PC (Program Counter) il valore contenuto in $4. Normalmente in tali locazioni si trova la Ram e quindi sembrerebbe che il 68000 sia costretto a saltare a caso. È qui che entra in gioco il memory overlay bit, che non fa altro che far corrispondere alla locazione 0 niente altro che la Rom del Kickstart. Questo avviene nell'Amiga 500 e 2000, nel 1000 invece il 68000 salta alla procedura di caricamento del Kickstart e poi al Kickstart stesso. La prima istruzione del Kickstart è una fatidica jmp $fc00d2, che eseguita in modo supervisore (in cui ci troviamo all'accensione) provoca un reset del sistema. Routine seguenti rimettono a posto l'overlay bit attivando quindi la Ram. A questo punto il sistema parte con tutte le varie inizializzazioni, Exec, librerie, controlli sui chip custom, sulla Ram, Autoconfigurazione, ecc.

Completate tutte queste inizializzazioni il sistema prova a leggere, sempre che non siano state attivate altre forme di Boot con il Kickstart 1.3, i blocchi 0 e 1 del Drive 0. Se non c'è alcun disco inserito, o non vi è un Boot Block valido viene visualizzata la fatidica mano che richiede il disco del Workbench ed il sistema continua all'infinito a tentare di leggere un Boot Block decente.

Boot Block validi

Abbiamo parlato di Boot Block valido. Questo perché il Boot Block ha una sua particolare struttura. Per prima cosa vediamo di capire quanto è lungo. Si sa che il Boot Block è contenuto nei blocchi 0 e 1 di un disco. I blocchi contengono 512 byte e quindi avremo a disposizione per il nostro Boot ben 1024 byte che sembrano pochini ma che ci consentono di fare un mucchio di cose. D'ora in avanti ci riferiremo al Boot Block non come a due settori di un disco, ma come ad una area di memoria di 1 KByte (che poi dovrà essere scritta sul disco a partire dal settore 0).

La prima cosa che il sistema andrà a controllare, dopo aver letto il Boot Block sarà la presenza (vedi fig. 1) di una prima longword di riconoscimento contenente quattro caratteri e precisamente la stringa «DOS» terminata da uno 0 (il C insegna). Successivamente controllerà se il checksum (del cui calcolo ci occuperemo più tardi) è corretto e se la terza long contiene il valore $00000370 che non è altro che il puntatore al primo blocco della directory.

Esaurita tutta questa parte di controllo il sistema lancerà il codice del Boot Block con una jsr all'indirizzo della quarta longword e cioè esattamente dodici byte più avanti dell'inizio della zona di memoria del Boot.

Per esempio supponiamo che il sistema abbia caricato il Boot Block a partire da $A0000 (ciò è perfettamente possibile per utenti di macchine con un Megabyte di Chip Ram). La prima operazione che verrà effettuata sarà il controllo della presenza del valore $444F5300 in $A0000, la seconda sarà il controllo della presenza del valore $00000370 in

$A0008 e la terza il controllo della validità del checksum contenuto in $A0004. Se tutte queste operazioni hanno successo il sistema salterà al Boot Block con una jsr $A000C e cioè esattamente dodici byte più avanti della prima locazione occupata dal primo byte del Boot Block, come dicevamo prima. In poche parole l'immagine in memoria sarà la seguente:

$A0000 - Stringa DOS/0
$A0004 - Checksum
$A0008 - Valore $00000370
$A000C - Inizio del codice

La presenza dei codici di controllo, di un checksum corretto e di un programma posto alla dodicesima locazione non è generalmente sufficiente alla corretta esecuzione di un Boot Block. Come viene specificato nel Rom Kernel Manual il KByte contenente il Boot Block viene caricato dal sistema in una posizione casuale della memoria. Ciò implica che il codice contenuto nel Boot Block sia rigorosamente rilocabile, il che non è una novità nel mondo di Amiga. Notate che per rilocabile intendo non il generico codice di Amiga, che viene compilato come se dovesse partire da $0 e poi rilocato dal Dos quando viene caricato, bensì codice PC relativo, che è in grado di gestire i suoi riferimenti interni in base al PC.

Chiariamo il tutto con un esempio. Se osservate la figura 2 noterete l'istruzione lea Dos(PC),a1. La sua funzione è quella di mettere in A1 l'indirizzo della stringa «dos.library» ponendo il PC in A1 e sommandogli un offset corrispondente alla distanza fra la locazione puntata dal PC e quella contenente la stringa. Se invece di lea Dos(PC),a1 avessimo scritto lea Dos,a1 il processore non farebbe altro che mettere in A1 l'offset della stringa «dos.library» riferito al primo byte del programma. In questo modo non avremmo problemi se caricassimo il codice a partire da $0, il che più che scarsamente probabile è pressoché impossibile. Risulta chiaro quindi che la semplice aggiunta di (PC) ci evita di preoccuparci della posizione in cui verrà caricato il nostro Boot Block eliminandoci una moltitudine di problemi. Lo scrivere codice PC relativo costa un piccolo impegno e la rinuncia a qualche istruzione, tuttavia ci consente di ottenere moduli che possiamo praticamente utilizzare ovunque.

A questo punto può esere utile notare che l'assemblatore standard Metacomco/TenchStar stampa la dicitura «Position Independent» se il codice compilato è PC relativo, «Relocatable» negli altri casi. Potete quindi usare questo sistema per controllare di avere scritto il codice come previsto.

Figura 1 - Questo è quello che potremmo vedere se disassemblassimo un Boot Block standard.

```
A0000   444F5300 00000000 00000370 'DOS........p'

A000C   43FA 0018 lea      $000A0026(pc),a1
A0010   4EAE FFA0 jsr      $FFA0(a6)
A0014   4A80      tst.l    d0
A0016   670A      beq.s    $000A0022
A0018   2040      movea.l  d0,a0
A001A   2068 0016 movea.l  $0016(a0),a0
A001E   7000      moveq    #$00,d0
A0020   4E75      rts
A0022   70FF      moveq    #$ff,d0
A0024   60FA      bra.s    $00A0020

A0026   646F732E 6C696272 61727900 'dos.library.'
```

Boot Block Standard

L'ultima cosa che ci rimane da specificare è la funzione del codice contenuto nel Boot Block. Un Boot Block standard, cioè quello creato dal comando Install non fa altro (vedi fig. 2) che trovare in memoria il modulo residente corrispondente alla Dos.Library e ritornare al sistema (attraverso una rts) lasciando in DO un opportuno codice che indica se l'operazione è stata eseguita con successo.

Ora che sappiamo tutto questo possiamo intervenire in due modi:

1) far precedere il codice standard da una nostra routine che può essere un semplice cambio di colore, un allocatore di memoria, una intro... o un virus... oppure

2) eliminare totalmente la routine di sistema per lanciare un nostro programma che prenderà il controllo totale dell'Amiga quale potrebbe essere il caricatore di un gioco, ecc.

Non considereremo per nulla la seconda ipotesi in quanto implica la stesura di un programma che gestisca direttamente l'hardware, senza usare le librerie e che non sarà mai in grado di far partire l'Amiga Dos se non attraverso un reset software o hardware del sistema. Si tratta di un metodo utilizzato da gran parte dei programmatori di videogame che hanno spesso problemi di velocità e di memoria in caso di convivenza con l'Amiga Dos e che quindi preferiscono bypassare il sistema per prendere il pieno controllo della macchina.

Il primo metodo invece è quello che più ci interessa in quanto ci consente di creare utility... utilissime di tutti i generi. Immaginiamo di avere un disco pieno e di volere infilarci a tutti i costi una intro o una semplice scritta in scroll. Basta scrivere un piccolo programmino lungo massimo 1 KByte e infilarlo nel Boot. Possiamo inserire prima del Boot un

```
;Codice sorgente per un Boot Block Standard
;Ricordarsi di calcolare il checksum dopo la compilazione

      dc.l $444F5300          ;Stringa DOS/0
      dc.l $00000000          ;Spazio per il checksum
      dc.l $00000370

      lea      Dos(pc),a1     ;Mette l'indirizzo della stringa "dos.library"
                              ;nel registro A1
      jsr      -96(a6)        ;Chiama FindResident
      tst.l    d0
      beq.s    Err            ;Non ha trovato il mod. resid.
      movea.l  d0,a0
      movea.l  $16(a0),a0
      moveq    #0,d0
Usc:  rts
Err:  moveq    #-1,d0         ;Setta il flag di errore
      bra.s    Usc

Dos:  dc.b 'dos.library',0    ;Stringa "dos.library"
```

Figura 2 - E questo è il sorgente di un Boot Block standard. Basta compilarlo, calcolare il checksum, e scriverlo su disco.

```
;Codice sorgente per un Boot Block personalizzato
;Ricordarsi di calcolare il checksum dopo la compilazione

      dc.l $444F5300          ;Stringa DOS/0
      dc.l $00000000          ;Spazio per il checksum
      dc.l $00000370

      move.w   #$00f0,$dff180 ;Cambia il colore di fondo
Lop:  btst     #6,$bfe001     ;Controlla se il mouse e' premuto
      bne.s    Lop            ;Se non lo  ritorna al loop

      lea      Dos(pc),a1     ;Mette  l'indirizzo  della stringa "dos.library"
                              ;nel registro A1
      jsr      -96(a6)        ;Chiama FindResident
      tst.l    d0
      beq.s    Err            ;Non ha trovato il mod. resid.
      movea.l  d0,a0
      movea.l  $16(a0),a0
      moveq    #0,d0
Usc:  rts
Err:  moveq    #-1,d0         ;Setta il flag di errore
      bra.s    Usc

Dos:  dc.b 'dos.library',0    ;Stringa "dos.library"
```

Figura 3 - Questo è il sorgente di un Boot Block personalizzato.

programma che alloca tutta la memoria Fast e non avremo più problemi con i vecchi programmi che necessitano della sola Chip Memory.

Se siamo cattivi possiamo anche infilare nel Boot Block un bel virus, una musichina, quello che più vi aggrada con l'unica limitazione di utilizzare solo la graphics library e accedere il più possibile all'Hardware per risparmiare spazio...

Come realizzare un Boot Block

Ovviamente non è questa la sede per pubblicare tutti gli esempi che ho citato poc'anzi per cui ci occuperemo della realizzazione di un semplicissimo Boot Block che non fa altro che cambiare il colore dello schermo e aspettare una pressione sul pulsante sinistro del mouse prima di continuare la procedura di Boot.

La routine che cambia il colore e attende la pressione del mouse è di una semplicità estrema (vedi fig. 3). In coda alla nostra routine metteremo la routine standard avendo cura di inserire in un posto accettabile la stringa «dos.library» utilizzata dal codice standard.

Per inserire il nostro programma nel Boot Block non dobbiamo fare altro che assemblare e linkare il codice, calcolare il checksum e trasferire il tutto su disco. Se possedete l'assemblatore standard Metacomco/TenchStar e il linker standard Lattice/Software Distillery basterà battere:

assem Boot.s -o Boot.o
blink Boot.o to-Boot

Ricordatevi assolutamente di inserire alla fine del vostro sorgente la riga
dcb.b 1024,0

definisce un blocco di 1024 byte contenente $00 che alloca una serie di un Kbyte di zeri dopo la fine del vostro codice. Ciò vi eviterà di trascinarvi su disco della memoria sporca e di avere problemi con il calcolo del Checksum.

Fatto tutto questo non resta che calcolare il checksum. Non dobbiamo fare altro che procurarci un monitor tipo il RossiMon e caricare il programma appena compilato con la funzione Load Segment (1 Boot). Il RossiMon risponderà con First Segment Loaded At $nnnnnn dove $nnnnnn corrisponde all'indirizzo del primo byte in cui è stato caricato il nostro Boot Block. Battete #$nnnnnn ed il RossiMon vi risponderà con Old: $00000000 New: $xxxxxxxx ove $xxxxxxxx è il checksum appena calcolato. Non vi resta che scrivere tutto su disco con >$nnnnnn 0 0 2 dove il primo 0 indica DF0:, il secondo 0 il settore di partenza, ed il 2 il numero di settori da scrivere. Se non possedete il RossiMon basta utilizzare i comandi corrispondenti, tuttavia potrebbe capitarvi un monitor che non possiede la funzione di calcolo del checksum. Se questo è il vostro caso leggete il prossimo paragrafo.

Il calcolo del checksum

Le prime volte che vedevo quei numeracci ero terrorizzato, anche perché la descrizione che dava il Rom Kernel Manual di quel checksum non era certamente molto chiara. Poi alla fine dopo alcuni tentativi ho scoperto l'arcano. Non si tratta d'altro che di partire dal valore $ffffffff e di sottrarre una dopo l'altra tutte le longword costituenti il Boot Block, ovviamente eccetto quella contenente il checksum. Non dovrebbe esservi difficile scrivere un programmino in grado di calcolare il checksum di un Boot Block, ma visto che siete sfaticati eccovi al soluzione:

```
move.l a0,a1
preserva il puntatore al Boot Block
clr.l $0004(a0)
cancella il valore del checksum
move.l #$ffffffff,d0
inizializza il contatore del checksum
move.w #$00ff,d1
inizializza il contatore delle longword; 256*4
= 1024 = 1 KByte

Rip: sub.l (a0)+,d0
dbf d1,Rip ;Hai finito ?
move.l d0,$0004(a1) ;Setta il checksum
rts
```

Questo programma non fa altro che calcolare il valore del checksum del Boot Block il cui indirizzo viene passato in A0 e scrivere tale valore in A0+4, cioè esattamente al posto in cui va inserito il checksum. Non vi rimane altro da fare che trasferire il tutto su disco.

Il controllo finale

Una volta scritto il Boot Block su disco provate a resettare il computer e ad effettuare un Boot con quel disco. Se non appare la manina e lo schermo rimarrà verde fino a quando premerete il tasto sinistro del Mouse per far partire la procedura di Boot tutto sarà andato alla perfezione... complimenti. Beh, mi pare ovvio che se premete il bottone sinistro del Mouse e il Boot non viene effettuato normalmente avete commesso qualche errore... per cui vi consiglio di rileggere l'articolo.

Troppo bello per essere vero

Sarebbe troppo bello se tutto finisse qui. C'è ancora una piccola particolarità che non abbiamo citato. Il Rom Kernel Manual dice che quando viene chiamato un Boot Block in A0 c'è il puntatore alla IORequestBlock utilizzata per caricare per l'appunto il Boot Block. Modificando a dovere questo IORequestBlock e chiamando la DoIO possiamo continuare a leggere dei settori da disco mediante la TrackDisk.Device. Questo ci serve in caso che il nostro Boot Block sia più lungo di un KByte. È in questa maniera che funzionano programmi quali il BootGirl e così via. Se non sapete come funziona l'IORequestBlock potete andarvi a leggere il Rom Kernel Manual e dare un'occhiata all'articolo sui device di MCmicrocomputer numero 88.

A risentirci. MC

GLI HARD DISK

FLASHBANK

Hard Disk e DMA controller su scheda per A2000 e Zorro Big Blue. Velocità trasferimento dati: 7,5 Mbit/sec. Si inserisce su slot a 100 pin. Autoboot con sistemi operativi 1.2 e 1.3. Autoconfing. Formattato con Fast File System.

FLASHBANK 32Mb 40ms L. 690000
FLASHBANK 60Mb 40ms L. 990000
FLASHBANK 60Mb 25ms L. 1240000

2 Modulo A2090 Autoboot

Modulo che rende autoboot i controller Commodore A2090. Si inserisce su uno slot a 100 pin. L. 129.000

MULTIBRAIN

Hard Disk e DMA controller per A500/1000. Autoboot. Autoconfing. Formattato con FFS. Espansione opzionale RAM da 2 a 8 MB.

MULTIBRAIN 20 Mb 40ms L. 890000
MULTIBRAIN 40 Mb 40ms L. 1090000
MULTIBRAIN 40 Mb 25ms L. 1390000
Modulo RAM 2Mb L. 690000 - 4Mb L. 1190000 - BMb L. 2440000

IMPACT A2000 GVP

HD controller SCSI più Espansione RAM 22Mb per A2000 o ZBB con autoboot. O Kb L. 440000 - 2Mb L. 990000

IMPACT HC2000

Come sopra, ma senza RAM, con la possibilità di montare l'hard disk direttamente su scheda. L. 410000

HARD DISK 20 MB 3.5'' 40ms L. 640000
HARD DISK 47 MB 3.5'' 28ms L. 890000
HARD DISK 80 MB 3.5'' 11ms L. 1890000

A2090A Commodore

Hd controller più Harddisk da 20 Mb per A2000. Autoboot. L. 1090000

A590 Commodore

Hd controller più hardisk da 20 Mb con espansione Ram da 0 a 2Mb autoboot per A500. L. 920000

HD2000card

Cobtroller ed HardDisk da su scheda per AMSTSRAD, IBM/XT o A2000 con Janus. HD 2000CARD 32MB L. 690000. 40MB L. 740000

Janus XT

Emulatore IBM/XT per A2000 + drive da 5,1/4 con garanzia Commodore Italia L. 840000

Janus AT

Emulatore IBM/AT per A2000 + drive da 5,1/4 con garanzia Commodore Italia. L. 1650000

LE ESPANSIONI DI MEMORIA

AMEGABOARD

Espansione di memoria per A500/1000 da 2 a 8Mb. Esterna. Autoconfigurante. Si installa sul connettore laterale. Munita di connettore passante per altre periferiche completa di LED e di interruttore per il diserimento senza disconnetterla dal computer. Dimensioni 21 x 10 x 4,7 cm. L. 890000

AMINTERAM

Espansione di memoria per A500 da 512 Kb. Si inserisce nell'apposito slot del computer. Con orologio e batteria tampone. L. 189000

SUPEROTTO HARDITAL

Espansione da 0-2-4-8Mb sulla stessa scheda per A2000 o ZBB. Completa di dislpay con dindicazione della memoria disponibile e di led di autoconfigurazione. Zeri wait state.

2Mb L. 740000. 4Mb L. 1240000. 8Mb L. 2240000

A2058 Commodore

Espansione da 2 a 8 Mb per A2000. 2 Mb L. 104000

KICKROM 1.3 A1000

Kickstart 1.3 su eprom senza saldature per A1000 con orologio tampone. Si inserisce sul connettore laterale del computer. Completo di connettore passante. L. 1490000

KICKROM 1.3 A500/A2000

Kickstart 1.3 su Eprom interno per A500/2000. Completo di deviatore per Kickstart 1.2. L. 89000

I DRIVE

ADRIVE

Drive da 3,5 esterno per A500/1000/2000. Con interruttore per il disinserimento e di connettore passante. L. 199000

ADRIVE TOWER

Come sopra ma triplo nello stesso contenitore. L. 490000

ADRIVE 2000

Drive interno da 3,5'' per A2000. L. 169000

ACCELERATORI-PROCESSORI-COPROCESSORI

ATTENZIONE!!! I PREZZI SOTTOINDICATI COMPRENDONO LE SCHEDE ACCELERATRICI SENZA PROCESSORI E COMPROCESSORI CHE SONO INDICATI A PARTE. QUESTO PER LASCIARE IL MASSIMO GRADO DI LIBERTA' ALL'UTENTE.

HURRICANE

Scheda acceleratirce per A1000/A2000. Hurricane A1000 L. 499000-
Hurricane A2000 L. 799000.

BANG

Scheda acceletrice per A1000/A500. L. 340000

HURRICANE MEMORY 1-4Mb

Espansione di memoria 32 bit per Hurricane. Hurricane Men 1MbL. 1190000

ADAPTER 030

Adattatore per 68030 per Hurricane e Bang. L. 24.000

PROCESSORI: 68010 L. 49000 - 68020 L. 29000 - 68030 L. 590000
COPROCESSORI: 68881 - 12MHz L. 240000 - 16MHz L. 290000
25MHz L. 740000 - 68882 16MHz L. 390000 - 25MHz L. 990000

A26260

Scheda acceleratrice contenente 68020 a 14,3MHz + 68881 a 16MHz + MMU68851 + RAM a 32 bit da 2 Mb L. 2190000

I DIGITALIZZATORI AUDIO VIDEO

DIGIBOARD

Digitalizzatore audio stereo più interfaccia MIDI per A500/1000/2000L. 99000

LIVE! ASQUARED

Digitalizzatore a colori video in tempo reale con effetti video per A500/1000 o A2000 (su scheda). Live 500 L. 549000 - LIVE 1000 L. 440000 - LIVE 2000 L. 630000

GENLOCK CARD A2301 Commodore

Scheda Genlock semiprofessionale per Amiga 2000. L. 390000

FLICKER FIXER

Scheda da inserire nello slot video dell'A2000 ed elimina il flicker. L. 690000.
Flicker Fixer + Monitor Multisync L. 1350000

ZORRO BIG BLUE

Chassis metallico per A500/1000 comprendente mainboard con 3slot 100 pin A2000 più 3 slot XT, 3 slot At compatibili più 1 slot CPU a 86 pin per schede con 68020/68881 (Hurricane, A2620). Completo di alimentatore switching da 180 W. Predisposizione per 2 drive da 3,5'', 1 - drive da 5,1/4'' 1 hard disk da 3,5'' o 5,1/4'' e digiboard. L. 470000

MODULO DRIVE

1 o 2 drive da 3,5'' -880Kb. L. 189000

MODULO MIDI + DIGI STEREO

scheda contenente digitalizztore stereo più interfaccia MIDI. L. 79000

ATTENZIONE!!! NELLO ZORRO BIG BLUE SI POSSONO MONTARE TUTTE LE SCHEDE PER L'AMIGA 2000 (JANUS XT/AT, AMEGADRIVE, A2058, SUPEROTTO, IMPACT, A2620, A2090, ecc.)

I MONITORS

COMMODORE 1084

Monitor HiRes per A500, A1000, A2000. L. 520000

PHILIPS 8833

Monitor stereo per Amiga o PC L. 520000

STAMPANTI

STAR LC 10 L. 420000
STAR LC 10 color L. 490000

STAR LC 24-10

Stampante a 24 aghi 150 cps NLQ. **L. 680000**

I COMPUTER

AMIGA 500

Completo di mouse e manuali. L. 720000
Come sopra ma con espansione da 1Mb e orologio L. 89.000

AMIGA 2000

CON MONITOR E SECONDO DRIVE DA 3.5'' L. 2250000
Come sopra più espansione da 2Mb L. 2860000
Come sopra più hard disk autoboot da 220Mb e espansione da 2Mb L. 3490000

DISCHETTI DA 3,5''

1 L. 1900 - 10 L. 1600 - 100 L. 1250

SONO INOLTRE DISPONIBILI TUTTI I COMPUTER E LE PERIFERICHE AMSTRAD. CHIEDERE.
DISPONIBILI ANCHE TUTTI I COMPATIBILI XT, AT E PS2. CHIEDERE

PER INFORMAZIONI E/O ORDINAZIONI

COMPUTER CENTER

VIA FORZE ARMATE 260 20152 MILANO
TELEFONO 02-4890213

VENDITA SOLO PER CORRISPONDENZA
TUTTI I PREZZI SONO IVA COMPRESA

FLASHBANK

- HARD DISC CARD PER A 2000 E ZORRO BIG BLUE
- AUTOBOOT CON EPROM
- FULL AUTOCONFIG
- CAPACITA: 20-32-40-60 Mb

MULTI BRAIN

- HARD DISC E CONTROLLER PER A 500 / A 1000
- AUTOBOOT CON EPROM
- AUTOCONFIG
- ESPANSIONE OPZIONALE DA 2 A 8 Mb
- 1 POSTO PER DRIVE DA 3,5
- MODULO OPZIONALE CON DISPLAY DELLA CAPACITÁ DI MEMORIA, CAPACITÁ HARD DISC E TRACCE DISC DRIVE
- CAPACITÁ: 20-32-40-60 Mb

SUPER 8

- ESPANSIONE DI MEMORIA PER A 2000 E ZORRO BIG BLUE
- AUTO CONFIGURANTE
- ZERO WAIT STATE
- DISPLAY CON INDICAZIONE DELLA CAPACITÁ INSTALLATA
- CAPACITÁ: 0-2-4-8 Mb

ZORRO BIG BLUE

- 3 SLOT A 100 PIN A 2000 COMPATIBILI
- 3 SLOT IBM XT COMPATIBILI
- 3 SLOT IBM AT COMPATIBILI
- 1 SLOT A 86 PIN PER 68020/68881
- 2 POSTI PER 2 DRIVE DA 3,5''
- 1 POSTO PER 1 DRIVE DA 5,1/4
- 1 POSTO PER HARD DISC
- ALIMENTATORE SWITCHING

HARDITAL

VIA TORTONA, 12
20144 MILANO
Tel. 02 - 8376887