di Pierluigi Panunzi

Indirizzi virtuali e indirizzi fisici

Nella scorsa puntata abbiamo imparato a conoscere due strutture fondamentali del software che ha per motore il microprocessore 80286: tali strutture, la «Local Descriptor Table» (LDT) e la «Global Descriptor Table» (GDT), sono fondamentali e rispecchiano l'importanza associata al concetto di «segmento», fatto che non abbiamo certo mancato di sottolineare e che anzi porremo ancora nel giusto risalto laddove sarà necessario.
Ricordiamo poi, in quanto di notevole importanza, che ogni segmento definito dal programma può essere di lunghezza variabile tra 1 e 64 k byte, consentendo cioè un'enorme flessibilità e modularità nella gestione dei programmi e soprattutto di sottoprogrammi, procedure e moduli in genere

Come si passa da un indirizzo virtuale ad uno fisico

La volta scorsa abbiamo interrotto il discorso subito dopo aver descritto i singoli elementi di una LDT e di una GDT, elementi detti «descriptor» in quanto «descrivono» appunto il relativo segmento, innanzitutto memorizzandone l'ampiezza e l'indirizzo fisico di partenza e poi indicandone i cosiddetti «Access rights» e cioè i «diritti di accesso», che rappresentano le modalità secondo cui è lecito accedere al segmento stesso: ora parleremo del meccanismo (abbastanza complesso) che consente di passare da un indirizzo virtuale all'effettivo indirizzo fisico.

Supponiamo dunque che in un certo momento la CPU deve fare riferimento ad una cella di memoria che il programmatore sa chiamarsi «ALFA».

Sappiamo dall'8086 che tale cella è posta all'interno di un segmento di dati individuabile dal valore corrente del registro DS.

Tanto per poter paragonare il meccanismo di conversione tra indirizzo virtuale e fisico dell'80286 con quello dell'8086, facciamo un piccolo passo indietro ricordando appunto il meccanismo (automatico, ovviamente) adottato nell'8086.

In questo caso conosciamo l'offset della cella ALFA, offset all'interno del Data Segment, per cui per trovarne l'effettivo indirizzo viene preso il contenuto del registro DS, viene moltiplicato per 16 (shiftato a sinistra di 4 bit) e al valore (a 20 bit) così ottenuto viene sommato il valore dell'offset.

In termini grafici si ha la «somma» di figura 1.

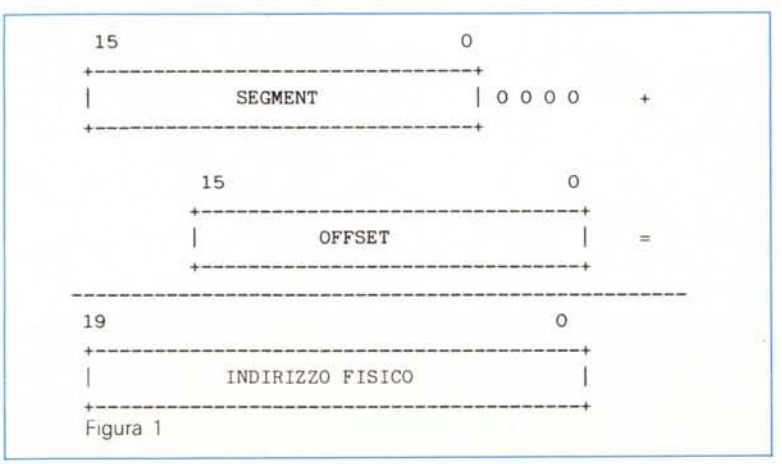


Figura 1

Invece nel caso dell'80286 ciò è molto più complicato e prevede una serie maggiore di «passi successivi», partendo sempre dal presupposto che l'offset della locazione è riferito al Data Segment.

Questa volta però dobbiamo ricordare che il registro DS non conterrà il valore del segmento di memoria a partire dal quale è posto il Data Segment, ma bensì contiene il valore di un «selector» e cioè l'indice che ha il «descriptor» di tale segmento all'interno della tabella dei descrittori.

Prima però di procedere oltre dobbiamo premettere altre considerazioni riguardanti l'«ambiente» in cui viene eseguito un certo programma, a sua volta strettamente legato al contenuto di particolari registri interni della CPU: una loro analisi ci aiuterà a comprendere i meccanismi di trasformazione di un indirizzo virtuale in uno fisico.

I registri GDTR e LDTR

Si tratta di una coppia di registri introdotti con il 286 e che consentono alla CPU di conoscere istante per istante l'«ambiente di lavoro» del programma corrente e cioè permettono di avere subito pronte delle informazioni riguardanti i segmenti utilizzati dal programma in esecuzione.

Sappiamo infatti che un generico modulo di programma ha le sue istruzioni in un proprio Code Segment, i suoi dati in un Data Segment ed eventualmente in un Extra Segment ed infine il suo stack in un proprio Stack Segment: per quanto detto nelle puntate precedenti, a meno di controindicazioni particolari, questo spazio di lavoro del programma, in breve l'«ambiente», è generalmente una «proprietà privata» del programma stesso, inaccessibile ed inattaccabile da processi esterni (a meno che ciò non

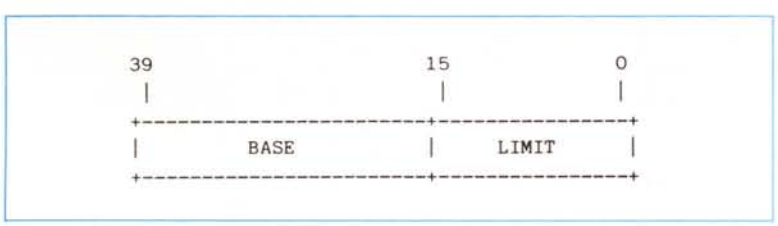
sia consentito «dall'alto»), in parole povere «locale».

In contrapposizione, sappiamo che ci possono viceversa essere degli «spazi di lavoro» al di fuori dell'ambito locale, vuoi perché si tratta di routine di sistema (condivise tra i vari processi e perciò presenti nel sistema «in singola copia», ma con accesso regolamentato), vuoi perché si tratta di dati anch'essi di «pubblico dominio», in entrambi i casi cioè presenti in un ambito «globale».

Ecco che dunque un processo in corso di esecuzione (che chiameremo, con terminologia corrente, «task»), farà sempre riferimento al suo ambiente locale nonché all'ambiente globale, grazie alla presenza dei due registri LDTR e GDTR, che costantemente mantengono informata la CPU riguardo i due ambienti gestiti dal task.

Per questo motivo infatti, subito prima di cedere il controllo ad un certo task, la CPU provvede ad inizializzare tale coppia di registri con dei valori che non sono altro che l'indirizzo fisico iniziale delle due «table», nonché la loro lunghezza: ma vediamo in particolare la struttura interna dei due registri, a cominciare dal GDTR.

Esso è formato da ben 40 bit, 24 dei quali (i più significativi) rappresentano l'indirizzo fisico iniziale della GDT, mentre i rimanenti 16 bit rappresentano l'ampiezza in byte della tavola stessa:



Invece, per quanto riguarda l'LDTR, il numero di bit sale a ben 56, in quanto in questo caso viene aggiunta una parte (più significativa) a 16 bit (figura 2).

Tale parte più significativa è a tutti gli effetti un «selector» (formato cioè dai campi INDEX, TI e RPL), ed in particolare è proprio il «puntatore» ad un elemento della GDT (infatti il campo TI deve valere "0") che è proprio il descrittore della LDT del processo: appunto in funzione di tale «selector», vengono caricati dalla CPU gli altri due campi, letti proprio in corrispondenza dell'elemento della GDT puntato.

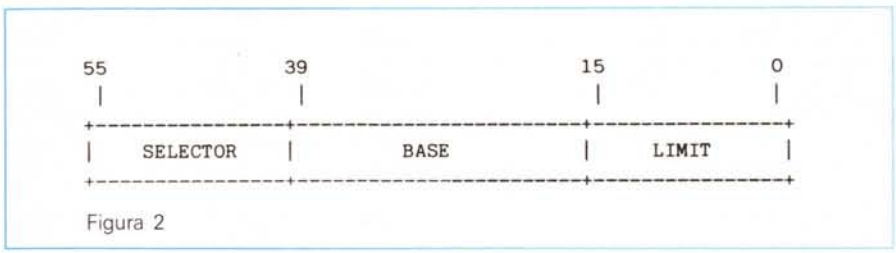


Figura 2

Per entrambi i registri esistono delle particolari istruzioni di caricamento e lettura (rispettivamente LGDT e SGDT per il GDTR e LLDT e SLDT per l'LDTR) che ovviamente possono essere eseguite solo a livello di privilegio più alto (quello del supervisore), data la fondamentale importanza delle informazioni in esse contenute.

In particolare con l'istruzione di caricamento del registro GDTR viene posto nei suoi sei byte (48 bit) il contenuto di sei byte consecutivi della memoria indirizzati dall'istruzione stessa: tali sei byte dovranno essere appunto i valori BASE e LIMIT della GDT in gestione al programma. Invece con l'istruzione LLDT si può caricare solo la parte più significativa del registro stesso (il campo SELECTOR), che sappiamo essere il selector di un elemento della GDT che a sua volta deve essere il descriptor di una LDT valida (quella di proprietà del task): se viceversa l'elemento puntato non è il descriptor di una LDT, come pure in altri casi di errore che nemmeno accenniamo, allora verrà generata un'apposita segnalazione di errore che inibirà l'esecuzione del comando richiesto.

Se invece tutto va bene, allora nei campi BASE e LIMIT verranno appunto memorizzati i valori che si trovano nell'elemento della GDT puntato dal campo SELECTOR.

Ora che abbiamo visto come sono costituiti i registri GDTR e LDTR, occupiamoci dei registri di segmento, che ci presenteranno alcune sorprese.

I registri di segmento

Non stiamo certo qui a ricordare quali e quanti sono i registri di segmento, dal momento che sono sempre i soliti e ben noti: quello che invece desideriamo segnalare è la loro effettiva struttura interna, completamente contrapposta alla struttura «esterna» e cioè visibile da parte del programma.

Abbiamo detto, fin dalla prima volta in cui abbiamo parlato dei registri di segmento, che con l'80286 tali registri contengono sempre e solo un «selector» e cioè, come oramai deve essere ben chiaro, il puntatore ad un elemento di una delle due «table», indifferentemente locale o globale, a seconda delle circostanze.

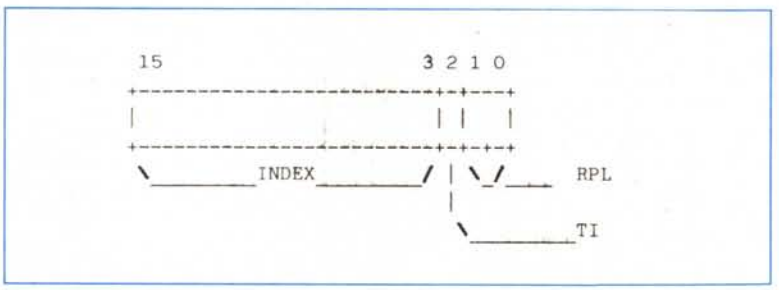
In particolare quando il supervisore cede il controllo al nostro task, avremo (come visto) all'interno dei registri LDTR e GDTR le informazioni che servono al nostro task per andare a cercare tutti i segmenti di cui ha bisogno.

A seguito di particolari meccanismi di cui parleremo nel seguito, subito prima che il task possa entrare in esecuzione,

verranno caricati i registri dei segmenti all'interno della CPU, fatto che perciò consente alla CPU stessa di sapere istante per istante dove andare a prendere le istruzioni da eseguire (CS), dove sono i dati su cui lavorare (DS ed ES) e dov'è lo stack (SS).

Il nostro task si troverà questi registri belli che pronti (altrimenti non potrebbe ovviamente andare in esecuzione!) e a sua discrezione (se ne avrà i privilegi) potrà cambiarne il contenuto, così come siamo abituati a fare in programmi dell'8088.

In particolare la struttura interna di tutti e quattro i registri di segmento (almeno la parte «visibile» e modificabile) è la seguente:



dove in base al valore del bit TI sapremo che il selector fa riferimento alla LDT (TI=1) oppure alla GDT (TI=0), mentre in base al valore del campo RPL potremo o meno effettuare determinate operazioni.

Quello che invece è completamente nuovo è il modo in cui è memorizzato all'interno della CPU il singolo registro di segmento: da questo punto di vista (e ciò accade molto spesso parlando del 286 e già i lettori se ne sono accorti strada facendo) ciò che è «visibile» dall'esterno non è altro che la classica punta dell'iceberg...

Infatti un Segment Register viene viceversa considerato dalla CPU come una mostruosità lunga ben 64 bit dei quali i 16 bit «visibili» sono i più significativi; in figura 3 vediamo in rappresentazione grafica da quali parti è costituito un Segment Register qualunque.

A guardar bene la figura, oltre al SELECTOR, ritroviamo degli elementi ben noti (l'«ACCESS RIGHTS BYTE», «BASE» e «LIMIT») che a pensarci bene fanno parte integrante di un «segment descriptor»: ed allora si può immaginare quello che succede in realtà quando il supervisore carica i registri di segmento oppure quando è il task a modificarli durante l'esecuzione.

Per i lettori con poca immaginazione vediamo dunque quali sono i meccanismi scatenati in queste due situazioni.

In particolare non appena in un programma viene trovata un'istruzione che effettua il caricamento diretto di un registro di segmento (MOV, LDS, LES oppure POP oppure ancora un salto «intersegment» e perciò subito prima dell'attivazione della nostra routine), il registro interessato viene normalmente caricato, ma stavolta viene effettivamente caricata la parte «SELECTOR» cioè quella «visibile».

Automaticamente invece la CPU provvede ad effettuare il caricamento degli altri campi con i corrispondenti campi del «segment descriptor» individuato dall'INDEX.

Ecco che perciò il valore del campo INDEX viene moltiplicato per 8 (shiftato a sinistra di 3 bit), proprio perché ogni elemento di una «descriptor table» è lunga 8 byte ed in base al valore del campo TI si andrà a leggere il corrispondente registro LDTR o GDTR: se si è nei limiti della rispettiva «table» e se poi, al solito, tutto è lecito, allora nei campi rimanenti del segment register verranno ricopiati i corrispondenti valori letti dall'elemento del segment descriptor, valori che così saranno disponibili alla CPU ed al programma stesso da quell'istante in poi.

Riflettiamo un istante su quanto abbiamo detto finora.

Subito prima dell'esecuzione del nostro task, ci ritroviamo dunque i registri «delle table» e quelli di segmento caricati con i valori corretti e relativi a quello che avevamo chiamato l'«ambiente di lavoro» di un programma: da questo punto in poi e cioè per tutto il tempo che è attivo il nostro task, questo potrà fare riferimento alle istruzioni del suo Code Segment, ai dati del suo Data Segment ed Extra Segment ed allo stack posto nello Stack Segment e perciò da ora in poi tutti i riferimenti ai quattro segmenti, con le dovute complicazioni del caso, rispecchiano quanto già sappiamo per averlo incontrato nell'8086.

Le complicazioni del caso riguardano il fatto che abbiamo sempre a che fare con un sistema che ha gran cura della «proprietà privata» e che perciò garantisce l'integrità delle informazioni relative ad un task da «attacchi» esterni: questo fatto verrà ripetuto fino alla fine...

Ci si può domandare poi quanto questa inizializzazione dei registri possa costare in termini di tempi di esecuzione: a parte il caricamento iniziale dei vari registri interni subito prima dell'attivazione del task, che rimandiamo alle prossime puntate, e a parte il caricamento dei registri LDTR e GDTR che avviene solo a livello di sistema, possiamo viceversa confrontare la differenza dei tempi di esecuzione di istruzioni che eseguono il caricamento di registri di segmento durante l'esecuzione del task, sia nel caso in cui si lavori in «Real Mode» che, appunto, in modo protetto.

Sappiamo che per caricare un registro di segmento abbiamo parecchie strade:
— il CS non può essere ovviamente caricato in modo diretto, ma solo indirettamente tramite una JMP o CALL «intersegment» oppure all'esecuzione

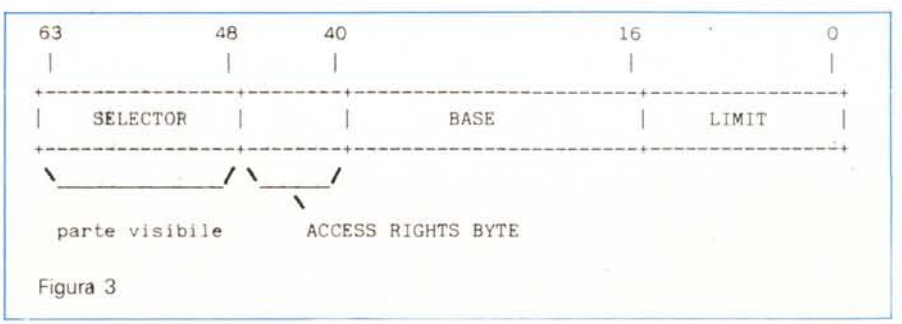


Figura 3

di una RET «intersegment» o di una IRET, rispettivamente al ritorno di una subroutine posta in un altro segmento ed al ritorno da una routine di gestione di un interrupt.
— I tre registri DS, ES ed SS viceversa possono essere caricati con semplici istruzioni di MOV (ma mai con valori «immediati», ma sempre tramite registri o locazioni di memoria), con il solito trucchetto di una PUSH seguita da una POP, oppure con un'istruzione particolare quale la LDS o la LES.

Nella tabellina seguente abbiamo posto, in corrispondenza delle istruzioni citate, dei valori relativi ai cicli di clock necessari all'esecuzione, relativi al caso in cui si lavori in «Real Mode»:

Real Mode	
Istruzione	**Cicli di clock**
JMP intersegment	11
CALL intersegment	13
RET intersegment	15
IRET	17
MOV xS,AX	2
POP xS	5
LDS memoria	7
LES memoria	7

Nel caso in cui si lavora in modo protetto, le cose cambiano leggermente: per le ultime quattro istruzioni si risente del fatto che ogni volta deve essere effettuato il caricamento completo del registro di segmento, per mezzo di accessi alla memoria, il che porta ai seguenti valori per i cicli di clock:

Protected mode	
Istruzione	**Cicli di clock**
MOV xS,AX	17
POP xS	20
LDS memoria	21
LES memoria	21

con tempi che perciò sono cresciuti rispettivamente di ben 8, 4, 3 e 3 volte.

Per quanto riguarda le prime quattro istruzioni il paragone non è così immediato: se ci si limita ad analizzare i tempi nel caso più favorevole si hanno i seguenti valori:

Protected mode	
Istruzione	**Cicli di clock**
JMP intersegment	23
CALL intersegment	26
RET intersegment	25
IRET	31

dove si vede che praticamente raddoppiano.

Ma questo è il caso più favorevole, in cui ad esempio si va da un segmento all'altro a parità di privilegi, senza passare attraverso «Task Gate» o «Call Gate», oppure ancora a seconda se si abbia o meno un «task switch»: no, non stiamo parlando arabo... sono solo alcuni dei «pochi» concetti dei quali parleremo nelle prossime puntate, concetti che sono, chi più chi meno, ostinatamente astrusi o vertiginosamente macchinosi.

Possiamo, a solo titolo di cronaca, riportare i valori «pessimi» per i tempi di esecuzione, valori che tengono conto di tutte quelle terminologie arabe di cui sopra:

Protected mode	
Istruzione	**Cicli di clock**
JMP intersegment	183
CALL intersegment	185
RET intersegment	55
IRET	169

lasciando ai lettori il banale calcolo di quanto aumentano (ricordiamo che si tratta dei casi pessimi).

Di questi tempi ci si deve ricordare tutte le volte che si deve cambiare uno dei quattro segmenti, fatto che perciò dovrebbe convincere i programmatori ad usare un solo Code Segment, un solo Stack e così via, a meno che non si tratti di applicazioni che sono «time independent». Chissà se nella versione multi-task del «Flight Simulator» il nostro fido Cessna 182 rallenta la sua velocità, sopraffatto da continui cambiamenti di task e conflitti di privilegi, mentre magari stiamo contemporaneamente gestendo i costi del nostro ufficio con un tabellone elettronico...

Un'istruzione innocua

All'inizio della puntata ci eravamo domandati come facesse il 286 a trasformare l'offset (puramente virtuale) della cella ALFA nel suo indirizzo fisico, effettivo: dopo aver introdotto un'altra tonnellata di concetti nuovi, ora siamo pronti a vedere il meccanismo, che ora non dovrebbe più sembrare complicato, che sta alla base della trasformazione «virtuale-fisico».

Supponendo dunque di incontrare nel nostro task l'istruzione.

MOV AX,ALFA

vediamo passo passo cosa fa il microprocessore:
— innanzitutto sa che deve far riferimento al DS in quanto si tratta di una locazione di memoria il cui contenuto deve mettere in AX.
— Legge il valore dell'offset contenuto nell'istruzione, valore che è, lo ricordiamo, l'offset rispetto all'inizio del segmento di dati corrente.
— Confronta tale valore con il valore LIMIT posto all'interno del registro DS (che ricordiamo essere formato dai campi SELECTOR, ACCESS RIGHTS, BASE e LIMIT) per verificare che risulti minore e cioè all'interno del segmento dati stesso, altrimenti viene generata la segnalazione di un tentativo di violazione.
— Verifica che l'accesso richiesto al dato sia di tipo lecito: nel caso del DS e dell'ES l'accesso può essere solo in lettura e/o scrittura ma non in esecuzione, mentre viceversa per il CS non può essere che di esecuzione e/o di lettura (ricordate che non si possono più scrivere codici automodificanti?!).
— Accede finalmente al dato sommando il valore dell'offset (16 bit) al valore BASE (24 bit) per ottenere dunque un indirizzo fisico a 24 bit. Ancora una volta deve apparire chiaro che al programma non è dato in alcun modo di sapere qual è l'indirizzo fisico corrispondente ad una locazione virtuale: a parte che anche se potesse conoscerlo (concedendo al task tutti i privilegi del caso), non potrebbe utilizzarlo praticamente in quanto abbiamo più volte ripetuto che in un ambiente multi-task la memoria (intesa come risorsa) viene assegnata istante per istante ad un determinato processo e perciò non è assolutamente detto che una certa cella sia assegnata sempre allo stesso task, così come accade nei mini-sistemi per non parlare dei grandi computer in cui potenti sistemi operativi continuano ad associare risorse ai singoli processi, cambiandole istante per istante a seconda delle esigenze del momento e secondo opportuni schemi di «scheduling».

Chiudiamo dunque la puntata non prima di aver fatto l'ultimo paragone sui tempi di esecuzione dell'istruzione.

MOV AX,ALFA

nel caso del modo «Reale» e «Protetto». Nel primo caso si hanno 5 cicli di clock, mentre in modo protetto, anche nei casi pessimi, si hanno... ancora 5 cicli di clock, così come tutto sommato era ben lecito attendersi, visto che tutti i tempi erano già stati consumati all'atto del «passaggio di consegne» al nostro task. Diamo appuntamento alla prossima puntata dove proseguiremo il nostro studio a cominciare dall'analisi dei livelli di privilegio. MC

AMIGA WORKSTATIONS GRAFICHE AMIGA

HARDWARE

AMIGA 500	930.000
AMIGA 500 + Monitor 1084	1.550.000
AMIGA 2000 senza monitor	1.950.000
AMIGA 2000 2 drive 3''1/2	2.190.000
ESPANSIONE 512K interna A500	Telef.
ESPANSIONE 1MB esterna A1000	Telef.
ESPANSIONE 2MB esterna A500/A1000	Telef.
ESPANSIONE 2MB interna A2000	Telef.
DISK DRIVE 3''1/2 esterno A500/A1000	290.000
DISK DRIVE 3''1/2 interno A2000	250.000
HARD DISCK 20MB EST. A500/A1000	1.250.000
HARD CARD 20MB SCSI A2000	1.250.000
HARD CARD 20MB 20MB SCSI A2000	750.000
HARD CARD 40MB MS-DOS A2000	950.000
Sistema a Cartridge da 12MB removibili della Kodak + 5 Cartridge (60 MB)	2.950.000
SCHEDA JANUS XT A2000	890.000
SCHEDA JANUS AT A2000	1.550.000
KIT SOSTITUZIONE MOTOROLA 68010	99.000
SCHEDA 68020 + 68881 16MHZ	1.850.000
AMIGA-EYE A500/A1000/A2000	130.000
VD AMIGA FRAMEGRABBER	750.000
VD 2000 DIGITALIZZATORE COLORE IN CVBS A500/A1000/A2000	1.150.000
TELECAMERA PANASONIC WV1410	750.000
TELECAMERA SECURIT T-979	550.000
STATIVO PROFESSIONALE 4 LAMPADE	350.000
AMIGA SOUND A500/A1000/A2000	150.000
INTERFACCIA MIDI A500/A1000/A2000	99.000
GENLOCK PROFESSIONALE	85.000
TAVOLETTE GRAFICHE KURTA:	
PENMOUSE (" × 9" 200 PPI	250.000
SERIE IS 8,5" × 11" 1000 PPI	790000
SERIE IS 12" × 12" 1000 PPI	990.000
SERIE IS 12" × 17" 1000 PPI	1.690.000
PENNA A DUE BOTTONI	290.000
CURSORE A 4 BOTTONI	290.000
CAVO E SOFTWARE PER AMIGA	110.000
STAMPANTI:	
PANASONIC KX-P1081 80 COL 120 CPS	550.000
NEC P2200 80 COL 216 CPS 24 AGHI	950.000
NEC P6 80COL 216CPS 245 AGHI	Telef.
NEC P6 KIT COLORE	Telef.
NEC P7 136 COL 216 CPS 24 AGHI	1.650.000
NEC P7 136 COL 216 CPS 24 AGHI	1.790.000
XEROX 4020 INK JET COLORE	3.450.000
OKI LASER LL6 PPM	3.850.000
XEROX 4020 INK JET COLORE	3.450.000
NEC LC 890 LASER POSTSCRIPT	Telef.
HARD COPIER SHINKO	Telef.
POLAROID PALETTE PER AMIGA	3.450.000

GENLOCK PROFESSIONALE

Caratteristiche tecniche:
- 2 ingressi video composito
- 1 ingresso RGB computer
- 1 uscita video composito
- 1 uscita radio/frequenza
- 1 uscita RGB + sinc.
- controllo e processo segnale video
- regolazione contrasto
- regolazifone colore
- regolazione saturazione
- mix ingressi 1 e 2
- esclusione video in/computer
- foratura sui colori RGB (croma-key)

OBLITERATOR

Gioco interattivo dalla grafica superba della Psygnosis inglese:
Impersona Drak, l'ultimo degli oblite-rators!

KURTA IS/ONE

Tavolette grafiche Kurta serie IS/ONE formati A4 (12" × 12") A3 (12" × 17")
Risoluzione di 1000 PPI
Accuratezza 0.035 INCH
Penna o cursore a croce a 4 bottoni
Di facile installazione, lavora con tutti i Packages grafici dell'Amiga

SOFTWARE ORIGINALE:

INFINITY SOFTWARE:	SHAKESPEARE	289.000**NEW!**
	GALILEO 2.0	89.000**NEW!**
ISM INC:	THE SURGEON	65.000
MICROSEARCH:	CITY DESK	189.000
MICROPROSE:	SILENT SERVICE	55.000
	MOEBIUS	49.000
	ULTIMA III	49.000
MICROMAGIC:	FORMS IN FLIGHT	110.000**NEW!**
MICROILLUSIONS:	FIRE POWER	35.000
	DYNAMIC CAD	690.000
	PROTON PAINT	380.000**NEW!**
MINDSCAPE:	DEFENDER OF THE CROWN	59.000
	HALLEY PROJECT	69.000
	DEJA VU	69.000
	UNINVITED	69.000
NEWTEK:	DIGI-PAINT	79.000
OXXY INC:	MAXIPLAN 500	190.000**NEW!**
	MAXIPLAN PLUS	250.000**NEW!**
PSYGNOSIS:	BARBARIAN	55.000**NEW!**
	OBLITERATOR	55.000**NEW!**
SUBLOGIC:	FLIGHT SIMULATOR	75.000
	JET	75.000
	SCENERY DISK 7	39.000
ZUMA:	TV SHOW	129.000
GOLD DISCK:	PROFESSIONAL PAGE	445.000NEW!
	PAGESETTER ITAL.	210.000**NEW!**
ACTIVISION:	HACKER II	29.500
	THE ART OF CHESS	29.500
	SHAGHAI	29.500
	BORROWED TIME	65.000
	LITTLE COMPUTER PEOPLE	35.000
	MINDSHADOW	35.000
	TASS TIMES	35.000
	PORTAL	55.000
	GEE BEE AIR RALLY	55.000**NEW!**
AEGIS:	ANIMATOR	175.000
	ARAZOK'S TOMB	49.000**NEW!**
	AUDIOMASTER	75.000**NEW!**
	DIGA	99.000**NEW!**
	DRAW PLUS	320.000
	IMPACT	110.000
	SONIX	99.000
	VIDEOTITLER	125.000**NEW!**
	PORT OF CALL	129.000**NEW!**
	VIDEOSCAPE 3D	299.000
BYTE BY BYTE:	SCULPT 3D	129.000**NEW!**
	ANIMATE 3D	199.000**NEW!**
COMMODORE:	MIND WALKER	69.00
	TEXTCRAFT PLUS	145.000**NEW!**
	SUPERBASE PERSONAL	190.000
	LIGISTIX	120.000
DISCOVERY:	ARKANOID	75.000**NEW!**
EPYX:	DESTROYER	29.000**NEW!**
	WINTER GAMES	29.000
	WORLD GAMES	29.000
NEW HORIZONS:	PROWRITE	175.000**NEW!**
NORTHEASTERN SOFT	PUBLISHER PLUS	129.000**NEW!**
RIGHT ANSWER GROUP:	THE DIRECTOR	89.000**NEW!**
METACOMCO:	MCC PASCAL	139.000
	ASSEMBLER LANGUAGE	139.000
EAGLE SOFTWARE:	BUTCHER 2.0	49.000**NEW!**
ELECTRONIC ARTS:	ADVENTURE C. SET	38.000
	ARTIC FOX	29.500
	BARD'S TALE I	29.500
	CHESSMASTER 2000	29.500
	INSTANT MUSIC	33.000
	MARBLE MADNESS	29.500
	SKYFOX	29.500
	TEST DRIVE	33.000**NEW!**
	DE LUXE MUSIC C.S.	94.000
	DE LUXE PAINT II	99.000
	DE LUXE PRINT	90.000
	DE LUXE VIDEO 1.2	109.000
	FERRARI FORMULA 1	38.000
	RETURN TO ATLANTIS	38.000
PROGRESSIVE P. &S	PIXMATE	94.000**NEW!**
MASTERTRONIC:	BLASTABAL	19.900**NEW!**
	FEUD	19.900**NEW!**
	KIKSTART II	19.900**NEW!**
	NINJA MISSION	19.900**NEW!**
	SPACE RANGER	19.900**NEW!**
FIREBIRD:	BUBBLE BOBBLE	29.000**NEW!**
MIRRORSOFT:	DARK CASTLE	49.000**NEW!**
	KING OF CHICAGO	59.000**NEW!**
	TETRIS	39.000**NEW!**
ANCO:	DEMOLITION	19.900**NEW!**
	FLIGHT PATH 737	19.900**NEW!**
	GRID START	19.900**NEW!**
	JUMP JET	19.900**NEW!**
	KARTING GRAND PRIX	19.900**NEW!**
	LAS VEGAS	19.900**NEW!**
	PHALANX	19.900**NEW!**
	SKY FIGHTER	29.000**NEW!**
	STRIP POKER	19.900**NEW!**
	THAI BOXING	19.900**NEW!**
	XR 35	19.900**NEW!**
RAINBIRD:	DRUM STUDIO	79.000**NEW!**
	GOLDEN PATH	79.000**NEW!**
	JINXTER	49.000**NEW!**
CDS:	FOOTBALL FORTUNE	49.000**NEW!**
MELBOURNE HOUSE:	ROADWARS	39.000**NEW!**
	XENON	39.000**NEW!**

TUTTI I PREZZI SONO IVA INCLUSA

PERSONAL COMPUTER

LINEA HITECH PERSONAL COMPUTER
LINEA XT 4.7/10 MHZ

XT-HT 256K 1FDD 360K TAST. AVANZ.	850.000
XT-HT 256K 2FDD 360K TAST. AVANZ.	1.050.000
XT-HT 256K 1FDD 360K HD 20MB TAST. AVANZ.	1.550.000
LINEA AT 10MHZ 0 WAIT STATE	
AT-HT 512K 1FDD 1.2MB TAST. AVANZ.	1.950.000
AT-HT 512K 1FDD 1.2MB 1 HD 20MB TAST. AVANZ.	2.550.000
AT-HT 512K 1FDD 1.2MB 1 HD 85MB TAST. AVANZ.	3.150.000
AT-HT 512K 1FDD 1.2MB 1 HD 140MB TAST. AVANZ.	4.750.000
LINEA 386 16-20 MHZ	
TOWER 2MB 1FDD 1.2 MB 1 HD 40MB TAST. AVANZ.	6.280.000
TOWER 2MB 1FDD 1.2MB 1 HD 85MB TAST. AVANZ.	7.750.000
TOWER 2MB 1FDD 1.2MB 1 HD 140MB TAST. AVANZ.	9.850.000
SCHEDE PC	
SCHEDA SERIALE	58.000
SCHEDA PARALLELA CENTRONICS	36.000
SCHEDA EGA AUTOSWITCH	490.000
SCHEDA FAX	1.450.000
SCHEDA COPY CARD II	160.000
HARD DISK	
HARD DISK 20MB + CONTROLLER	590.000
HARD DISK 40MB + CONTROLLER	950.000
HARD CARD 20MB	690.000
HARD CARD 40MB	1.050.000
COPROCESSORI MATEMATICI	
INTEL 8087 6MHZ	250.000
INTEL 8087 8MHZ	380.000
INTEL 80287 6MHZ	390.000
INTEL 80287 8MHZ	580.000
INTEL 80287 10MHZ	690.000
INTEL 80387 16MHZ	1.250.000
MONITOR	
PHILIPS 7502/7513 MONOCROMATICO 12" 180.000	
PHILIPS 9073 EGA COLORE 14"	850.000
PHILIPS 8833 COLORE 14"	550.000
MULTISYNC MONOCROMATICO	550.000
MULTISYNC COLORE	1.250.000
MODEM	
ESSEGI 1200M 300/1200 BAUD V21/V22 FULL DUPLEX	360.000
ESSEGI 1203M 300/1200/75 V21/V23 VIDEOTEL	420.000
ESSEGI 2400M 1200/2400 BAUD V22/V22 BIS	750.000
ESSEGI 1200C CARD	360.000
TELEFAX	
TELEFAX BACON-TELEFONO G2/G3 FORMATO A4	2.250.000

VENDITA PER CONTRASSEGNO SU TUTTO IL TERRITORIO NAZIONALE. OFFERTE E PREVENTIVI SU WORKSTATIONS GRAFICHE COMPLETE. SETTORI CAD 2D/CAD 3D/ANIMAZIONI 3D/DIGITALIZZAZIONI/VIDEO BROADCAST/DESKTOP PUBLISHING.
SI INVIANO A RICHIESTA SCHEDE TECNICHE PRODOTTI.
SCONTI PER RIVENDITORI QUALIFICATI.

Pix computer

PIX COMPUTER S.R.L.
VIA F. D'OVIDIO, 6c
TEL. 06/8293607-825731
00137 ROMA
COMPUTER & Co.
P. IVA 08309630583