di Fabio Schiattarella

Il linguaggio macchina sullo Spectrum (seconda parte)

Nella puntata precedente abbiamo visto come si utilizza la funzione USR, che permette di lanciare un programma in L/M ed inoltre restituisce il valore contenuto nella coppia di registri BC al momento del rientro al Basic. Sfortunatamente questa è l'unica possibilità di scambio di parametri che i progettisti della Sinclair Research hanno pensato di introdurre nel Basic. Nulla di specifico è stato introdotto per quanto riguarda il passaggio di più di un valore, ma neanche per l'operazione opposta, ovvero il passaggio anche di un singolo parametro dal programma Basic a quello in linguaggio macchina. Vediamo che cosa si può fare per ovviare a queste mancanze.

Un'idea assai semplice, ma che si dimostra valida in moltissimi casi, è quella di usare le posizioni di memoria come se si trattasse di una sorta di casella postale.

Se due persone comunicano attraverso il fermoposta, esse si scambiano delle informazioni senza che una conosca l'indirizzo dell'altra. L'informazione che esse condividono è un indirizzo comune cui fare riferimento.

Noi vogliamo realizzare un meccanismo esattamente analogo, in cui l'unico tramite tra programma Basic e programma in linguaggio macchina è un insieme di locazioni di memoria il cui indirizzo è conosciuto da entrambi. Vediamo subito di mettere in pratica questa idea in un caso molto semplice. Consideriamo un programma che esegua bit per bit una operazione logica fondamentale (and, or, xor, not) su un intero byte.

Nel Basic dello Spectrum sono presenti le operazioni di AND, OR e NOT, non si tratta però di operatori logici nel vero senso della parola, ma di operatori aritmetici: il principio è che viene considerato come vero un qualsiasi valore diverso da zero, come falso lo zero. Anche nel risultato lo ZXBasic differisce dallo standard. L'AND tra due valori per esempio restituisce 0 se uno dei due è zero, oppure il primo dei due valori se entrambi sono diversi da zero. L'OR si comporta in maniera piuttosto curiosa. Se entrambi i valori sono zero, restituisce regolarmente il risulato 0. Se uno solo dei valori è zero, il risultato è pari all'altro argomento, ovvero un intero. Se entrambi gli argomenti sono diversi da zero, il risultato è 1. Il NOT accetta un unico argomento e restituisce come valore 1 o 0 a seconda che il numero sia diverso o uguale a zero.

Sebbene tutto ciò funzioni egregiamente, dovendo scrivere espressioni condizionali come quelle mostrate nel capitolo 13 del manuale d'uso, talvolta può essere assai più comodo avere a disposizione operazioni che anziché operare sui numeri intesi come interi, lavorino bit a bit sulla loro rappresentazione. In particolare risulta spesso utile disporre di operatori logici che operino sui byte. Per esempio

51 OR 21 = 55

Vista così una tale espressione non dice granché, però riscrivendo i tre numeri in colonna nella loro rappresentazione binaria

```
00110011 OR
00010101 =
--------
00110111
```

possiamo verificare che il risultato contiene un uno binario ogni volta che la cifra corrispondente in almeno uno dei due operandi è pari ad uno.

Operatori logici su singoli byte sono disponibili nel linguaggio macchina dello Z80, ed è estremamente semplice realizzare un programmino richiamabile da Basic che presa una coppia di byte ne faccia l' and, l' or o l' or esclusivo bit a bit e restituisca il risultato. Nell'esempio che vedete svolto in figura 1, ci siamo limitati ai casi di and e not. Con la semplice sostituzione del codice relativo all'operazione AND, è possibile ottenere anche le funzioni OR ed XOR (or esclusivo). I codici corrispondenti alle operazioni sono:

NOME OPERAZIONE	CODICE ESADECIMALE	CODICE DECIMALE
AND B	A0	160
OR B	B0	176
XOR B	A8	168
CMA	2F	47

le prime tre svolgono l'operazione usando come operandi il contenuto dei registri A e B, e ponendo il risultato in A, la CMP complementa il contenuto dell'accumulatore A, ossia cambia gli zeri in uni e viceversa bit a bit.

Il programma quindi si limita a leggere gli operandi nelle locazioni di memoria usate come caselle postali, a porli nei registri A e B, a calcolare il risulato e a copiarlo nell'apposita locazione di memoria dove il programma Basic lo andrà a rileggere. In alternativa si potrebbe copiare il risultato

Figura 1

```
        AND DI DUE BYTE

  5B00              10 PRIMO  EQU   23296
  5B01              20 SECON  EQU   23297
  5B02              30 TERZO  EQU   23298
  5B04              40        ORG   23300
  5B04 3A005B       50        LD    A,(PRIMO)
  5B07 47           60        LD    B,A
  5B08 3A015B       70        LD    A,(SECON)
  5B0B A0           80        AND   B
  5B0C 32025B       90        LD    (TERZO),A
  5B0F C9          100        RET


   1 REM AND DI DUE BYTE
  10 FOR i=23300 TO 23311
  20 READ b: POKE i,b
  30 NEXT i
  40 DATA 58,0,91,71,58,1,91,160,50,2,91,201
  50 INPUT "primo valore ";b: IF b>255 THEN GO TO 50
  60 POKE 23296,b
  70 INPUT "secondo valore ";b: IF b>255 THEN GO TO 70
  80 POKE 23297,b
  90 RANDOMIZE USR 23300
 100 PRINT "AND= ";PEEK 23298
```

```
        NOT DI UN BYTE

  5B00              10 SCAMBI  EQU  23296 ; locazione per
                                            lo scambio dei
                                            valori
  5B04              20         ORG  23300
  5B04 3A005B       30         LD   A,(SCAMBI)
  5B07 2F           40         CPL  ; esegue operazione not
  5B08 32005B       50         LD   (SCAMBI),A
  5B0B C9           60         RET


   1 REM NOT DI UN  BYTE
  10 FOR i=23300 TO 23307
  20 READ b: POKE i,b
  30 NEXT i
  40 DATA 58,0,91,47,50,0,91,201
  50 INPUT "valore ";b: IF b>255 THEN GO TO 50
  60 POKE 23296,b
  90 RANDOMIZE USR 23300
 100 PRINT "NOT= ";PEEK 23296
```

Figura 2

```
        SOMMA DI INTERI A 16 BIT
5CB0            10 SOMMA  EQU  23728 : coppia di locazioni
                                       non utilizzate
                                       dal sistema
5B00            20 PRIMO  EQU  23296
5B02            30 SECON  EQU  23298
5B04            40        ORG  23300
5B04 2A005B     50        LD   HL,(PRIMO)
5B07 ED5B025B   60        LD   DE,(SECON)
5B0B 19         70        ADD  HL,DE : il risultato
                                       e' in HL
5B0C 22B05C     80        LD   (SOMMA),HL
5B0F C9         90        RET
```

```
  1 REM SOMMA DI DUE INTERI
  2 REM A SEDICI BIT
 10 FOR i=23300 TO 23311
 20 READ b: POKE i,b
 30 NEXT i
 40 DATA 42,0,91,237,91,2,91
 45 DATA 25,34,176,92,201
 50 INPUT "primo valore ";b: IF b>32767 THEN GO TO 50
 60 POKE 23296,b-INT (b/256)*256
 70 POKE 23297,INT (b/256)
 80 INPUT "secondo valore ";b: IF b>32767 THEN GO TO 80
 90 POKE 23298,b-INT (b/256)*256
100 POKE 23299,INT (b/256)
110 RANDOMIZE USR 23300
120 PRINT PEEK 23728+(256*PEEK 23729)
```

Figura 3

```
        AND A 16 BIT

5C0B            10 DEFADD EQU  23563
5B00            20        ORG  23296
5B00 DD2A0B5C   30        LD   IX,(DEFADD) : carica puntatore
                                             nel registro base
5B04 DD7E04     40        LD   A,(IX+4) : calcola byte meno
                                          significativo
5B07 DD560C     50        LD   D,(IX+12)
5B0A A2         60        AND  D
5B0B 4F         70        LD   C,A
5B0C DD7E05     80        LD   A,(IX+5) : calcola byte piu'
                                          significativo
5B0F DD560D     90        LD   D,(IX+13)
5B12 A2        100        AND  D
5B13 47        110        LD   B,A
5B14 C9        120        RET
```

```
 1 REM  AND A 16 BIT CON DEF FN
 2 DEF FN a(x,y)=USR 23296
10 FOR i=23296 TO 23316
20 READ b: POKE i,b
30 NEXT i
35 DATA 221,42,11,92
40 DATA 221,126,4,221,86,12,162,79
45 DATA 221,126,5,221,86,13,162,71,201
50 INPUT "primo valore ";c: IF c>65535 THEN GO TO 50
80 INPUT "secondo valore ";b: IF b>65535 THEN GO TO 80
90 PRINT FN a(c,b)
```

nella coppia di registri BC, cosicché esso venga restituito direttamente tramite la funzione USR.

Il programa Basic legge e scrive nelle apposite locazioni di memoria mediante delle PEEK e delle POKE. Virtualmente non esistono limitazioni riguardo al numero e alla posizione delle celle da utilizzare per il passaggio dei parametri, purché naturalmente non si vada a scrivere sopra ad altre informazioni utili, o si cancelli una parte del programma.

È possibile utilizzare come area di riferimento il buffer della stampante, come già abbiamo fatto più volte. Nel caso che due soli byte siano sufficienti potete usare una coppia di locazioni di memoria rimaste inutilizzate all'interno dell'area delle variabili di sistema. Si tratta della locazione di indirizzo 23728 e della successiva.

Questo metodo per passare i parametri è sufficientemente generale e riesce a coprire praticamente ogni esigenza del programmatore. Non altrettanto bene si può dire riguardo alla sua eleganza e concisione. Ogni volta che viene chiamato un sottoprogramma in linguaggio macchina sono necessarie diverse istruzioni Basic: una che contenga la USR, e almeno una POKE per ogni parametro che deve essere passato. Analogamente dopo la chiamata sono necessarie almeno tante istruzioni PEEK quanti sono i parametri che il programma in linguaggio macchina deve restituire al programma chiamante. Il discorso si complica parecchio se i valori da passare da un programma all'altro sono interi maggiori di 255. In questo caso un singolo byte non è più sufficiente a contenere il dato, e una istruzione POKE o PEEK non basta a scrivere o a leggere un intero dato.

Vediamo quindi come si fa a copiare in una coppia di locazioni di memoria adiacenti, in un formato che sia accettabile da parte di un programma in linguaggio macchina, il valore di una variabile minore di 65535.

Indichiamo con <ind> l'indirizzo della prima delle due celle destinate a contenere il numero. Supponiamo che il valore da copiare sia contenuto nella variabile N. Per trasferire tale valore nelle due locazioni <ind> e <ind>+1 sono necessarie le seguenti due istruzioni

```
POKE <ind>,N-INT(N/256)
POKE <ind>+1,INT(N/256)
```

per il passaggio inverso, ovvero per copiare in una variabile un intero a 16 bit:

```
LET N = PEEK <ind> + (256 * PEEK (<ind> + 1))
```

sostituendo ad <ind> il valore numerico di un indirizzo vi accorgerete che l'operazione è molto meno complicata di quanto possa sembrare.

Avrete probabilmente notato come nel trasferire il numero abbiamo effettuato una inversione. Abbiamo posto il byte con la parte più significativa del numero, quella con le migliaia e le decine di migliaia, dopo il byte con la parte meno significativa, quella con le unità e le decine. Abbiamo seguito l'ordine inverso a quello abituale in cui le cifre più "pesanti" vengono prima. Non si tratta di un errore, si è dovuta seguire questa procedura perché questo è il modo in cui il microprocessore si aspetta che siano memorizzati i valori a sedici bit. Quando con una istruzione di load carichiamo il numero in una coppia di registri le cose vengono rimesse a posto. Ad esempio l'istruzione

```
LD HL,(nn)
```

il cui codice esadecimale è 2A (decimale 42) carica in H il contenuto della locazione <nn>+1 (parte più significativa) ed in L il contenuto di <nn> (parte meno significativa). Notate per inciso come lo stesso indirizzo nn sia pure esso fornito secondo l'ordine: byte meno significativo, byte più significativo.

Tutto ciò ingenera spesso ad una certa confusione e può portare il programmatore a commettere diversi errori, se egli provvede da solo alla generazione del codice macchina. Un assemblatore al contrario provvede automaticamente a compiere le inversioni necessarie e accetta le costanti a sedici bit nel consueto ordine byte più significativo, byte meno significativo.

Ovviamente uno scambio analogo a quello appena descritto deve essere effettuato quando si effettua l'operazione di codifica inversa, ovvero per copiare in una variabile una costante a 16 bit prodotta dal programma in codice macchina.

Come semplice esempio consideriamo il programma di figura 2, che effettua la somma di due interi compresi tra 0 e 32767. Una tale limitazione è stata imposta per evitare eventuali problemi di trabocco, nel caso peggiore abbiamo infatti: 32767 + 32767 = 65534. I valori vengono passati mediante tre coppie di locazioni consecutive poste all'inizio del buffer della stampante.

Un metodo assai elegante per lanciare un programma in linguaggio macchina e passargli un qualsiasi numero di parametri, lo abbiamo trovato descritto in un articolo di Mike James apparso sul numero del maggio 1984 della rivista inglese "Electronics & computing monthly". Usando tale metodo un programma in linguaggio macchina può essere lanciato mediante la chiamata di una funzione predefinita dall'utente. Tutti i parametri che il programa Basic deve passare al programma in linguaggio macchina possono essere trasmessi direttamente come argomenti della funzione stes-

Figura 4

```
           NOT A 16 BIT

5C0B            10 DEFADD EQU   23563
5B00            20        ORG   23296
5B00 DD2A0B5C   30        LD    IX,(DEFADD)  ; carica puntatore
                                               nel registro base
5B04 DD7E04     40        LD    A,(IX+4)  ; calcola byte meno
                                            significativo
5B07 2F         60        CPL
5B08 4F         70        LD    C,A
5B09 DD7E05     80        LD    A,(IX+5)  ; calcola byte piu'
                                            significativo
5B0C 2F        100        CPL
5B0D 47        110        LD    B,A
5B0E C9        120        RET
```

```
 1 REM  NOT A 16 BIT CON DEF FN
 2 DEF FN n(x)=USR 23296
10 FOR i=23296 TO 23310
20 READ b: POKE i,b
30 NEXT i
35 DATA 221,42,11,92
40 DATA 221,126,4,47,79
45 DATA 221,126,5,47,71,201
50 INPUT "valore ";c: IF c>65535 THEN GO TO 50
90 PRINT FN n(c)
```

sa. Ciò comporta molteplici vantaggi di eleganza, leggibilità e risparmio nel numero di istruzioni. Inoltre il tutto si svolge in maniera completamente trasparente per chi programma in Basic, ovvero non ci si deve assolutamente preoccupare di come i valori vengano trasmessi al programma in linguaggio macchina.

Il metodo si basa sul particolare modo con cui vengono gestite la definizione e la chiamata di funzioni da parte del Basic dello Spectrum. Quando viene definita una funzione mediante una DEF FN, il sistema oltre a generare il codice relativo a tale istruzione provvede a riservare una certa quantità di spazio nel corpo del programma. Per ogni variabile numerica vengono riservati cinque byte nell'area di memoria occupata dal programma. Ad esempio data l'istruzione

10 DEF FN a(x,y) = x + y

essa verrebbe codificata nella seguente maniera:

Codifica significato

- 00 0A due byte che contengono il numero di linea
- 18 00 due byte che indicano la lunghezza della linea
- CE il codice dell'istruzione DEF FN
- 61 il codice ASCII della lettera "a", nome della funzione
- 28 codice ASCII di "("
- 78 codice ASCII di x, primo argomento (N.B. la variabile DEFADD punterà qui)
- 0E serve a specificare che ciò che segue rappresenta un valore numerico a cinque byte

seguono cinque byte per contenere il dato,

- 2C codice ASCII per ","
- 79 codice ASCII per "y"
- 0E specifica che segue un valore numerico a cinque byte

seguono cinque byte per contenere il dato,

- 29 codice ASCII per ")"

segue il resto della rappresentazione.

Quando durante il corso del programma, la funzione viene chiamata mediante uno statement FN, ad esempio nel nostro caso:

LET C = FNa(D+5)

i valori da passare alla funzione "a" non vengono posti nell'area destinata alle variabili, come avviene per tutte le altre variabili, ma vengono ricopiati direttamente nei byte liberi che erano stati riservati nel corpo stesso del programma all' atto della definizione della funzione. La variabile di sistema DEFADD in seguito alla chiamata con FN, punta al primo parametro nel corpo della definizione della funzione. È facile quindi, conoscendo l'indirizzo specificato in DEFADD e la posizione relativa dei valori nel corpo della definizione della funzione, andare a recuperare i singoli valori passati dal programma chiamante.

Nell'esempio che abbiamo visto sopra i cinque byte relativi al primo parametro trasmesso cominciano all'indirizzo

(DEFADD) +2 = 2 + il valore contenuto in DEFADD

e quelli relativi al secondo parametro cominciano all'indirizzo

(DEFADD) + 10 = 10 + il valore contenuto in DEFADD.

I valori numerici vengono memorizzati secondo il consueto formato a cinque byte, come illustrato nel capitolo 24 a pagina 125 del manuale (edizione inglese). In particolare se si tratta di interi compresi tra -65535 e 65535, il valore assoluto del numero viene memorizzato nel terzo e quarto dei cinque byte, secondo l'ordine byte meno significativo, byte più significativo. Di conseguenza tali valori a sedici bit saranno memorizzati nelle coppie di locazioni di indirizzo

(DEFADD) + 4 la prima
(DEFADD) + 12 la seconda

Vediamo come è possibile sfruttare tutto ciò per i nostri scopi. L'idea è semplice: dato che la USR, che serve a lanciare i programmi in linguaggio macchina, è una funzione numerica come tutte le altre, perché non inserirla nel corpo di una definizione di funzione?

Per esempio

DEF FN a(x,y) = USR <indirizzo>

In questo modo ogni volta che richiamiamo la funzione, ad esempio

LET n = FN a(2345,32000)

la sequenza delle operazioni è la seguente: la variabile di sistema DEFADD viene fatta puntare al nome del primo parametro che deve essere ricopiato nel corpo della funzione, tutti i parametri vengono trasferiti nel corpo della funzione (naturalmente se al posto di una costante od una variabile ci fosse una espressione più complessa, questa verrebbe prima calcolata), viene lanciato il programma in linguaggio macchina a partire dall'indirizzo specificato all'atto della definizione della funzione nella USR. Quindi la variabile di sistema DEFADD, nel momento in cui viene lanciato il programma in linguaggio macchina, punta effettivamente all'inizio dell'area in cui sono memorizzati i parametri. È abbastanza facile, nell'ambito del programma in linguaggio macchina, andare a recuperare i valori che vengono trasferiti come parametri della FNa, basta sommare al contenuto corrente di DEFADD delle quantità opportune.

Nel caso dell'esempio visto sopra, le istruzioni per andare a recuperare la coppia di valori a sedici bit corrispondenti alle variabili x e y potrebbero essere le seguenti:

```
LD IX,(DEFADD)
LD B,(IX+5)
LD C,(IX+4)
LD C,(IX+13)
LD E,(IX+12)
```

A questo punto possiamo riprendere l'esempio degli operatori logici AND, OR, NOT trattato all'inizio e svilupparlo secondo la metodologia appena introdotta. Questa volta ci estenderemo al caso di valori lunghi fino a sedici bit.

Si tratta quindi di definire la funzione

DEF FN 1(x,y) = USR 23296

dove 23296 è l'indirizzo iniziale del buffer della stampante in cui verrà allocato il programma in linguaggio macchina. Se tale programma conservasse il risultato finale nella coppia di registri BC la chiamata potrebbe avvenire semplicemente per mezzo dell'istruzione

PRINT FN a(BIN 111111, BIN 101010)

che restituisce come valore 42, la cui codifica binaria è proprio 101010. Notate come tutta l'operazione si svolga senza che l'utente si debba minimamente preoccupare di come venga effettuata la chiamata del programma in linguaggio macchina e di come vengano passati i parametri ad esso.

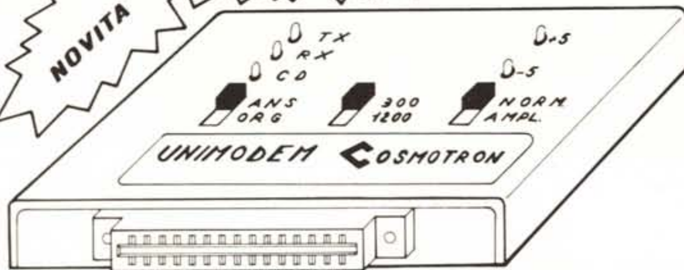
Il programma che effettua l'and logico a sedici bit, cui ci siamo appena riferiti, è illustrato in figura 4. Effettuando semplici modifiche, analoghe a quelle relative al programma di figura 1, è possibile realizzare le operazioni di or e not, che possono risultare utili in moltissime occasioni, specialmente quando si ha a che fare con l'hardware della macchina e sono necessarie operazioni sui singoli bit delle celle di memoria. Ad esempio per modificare gli attributi dello schermo (colore, sfondo, luminosità, lampeggio, inverso) può essere necessario accendere o spegnere singoli bit all'interno della mappa dello schermo in memoria. L'argomento è assai interessante e lo tratteremo in una prossima puntata di TuttoSpectrum.

PER CBM-64 (*)

L. 299.000

IVA COMPRESA!!

(*) Commodore Business Machines
Trade Mark



COMUNICATE CON UNIMODEM

CARATTERISTICHE TECNICHE:

- Modem con accoppiatore acustico
- 300baud in Half/Full Duplex
- 1200 baud in Half Duplex
- modo Answer e/o Originate
- segnale di uscita del modem può essere amplificato di circa 10 dB
- si applica alla USER PORT del CBM-64
- non necessita di alimentazione esterna
- tre led segnalano la presenza della portante, dei dati in trasmissione (TX) ed in ricezione (RX).
- UNIMODEM è fornito con un manuale ed un disco contenente programmi per trasmettere e/o ricevere files di testo o files Basic con opportuno programma 'TOKENIZZATORE'!!.
- UNIMODEM è corredato con 'speciali cuffie' che si adattano senza difficoltà a qualsiasi tipo di cornetta telefonica.

Programmatore di EPROM da 2K fino a 32K Bytes!!!

PER CBM-64 (*)

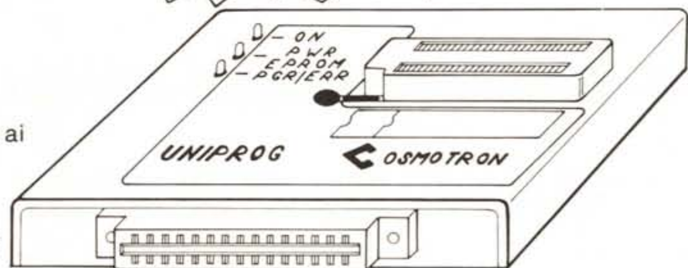
PROGRAMMATE CON UNIPROG

L. 299.000

IVA COMPRESA!!

UNIPROG è fornito con:

- manuale di uso con documentazione supplementare
- schedina per due EPROM di tipo 2764/32 allocabili in $8000 - $9FFF ed in $A000 - $BFFF
- **disco con i seguenti programmi:**
  - UNIPROG che gestisce il programmatore e si autoriloca al top della memoria di utente
  - PROG. AUTO-START per far eseguire un auto-start ai V/s programmi che allocherete a partire da $8000
  - UNIPROG 2.0 BOOT è il caricatore del seguente programma
  - UNIPROG C6D0-CFFF è il programma che gestisce UNIPROG lasciando la memoria di utente libera.

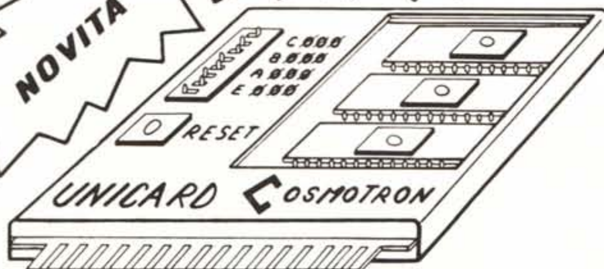


- UNIPROG non necessita di alimentazione esterna, si collega alla user port, non ha alcun interruttore perché è controllato con software di gestione linkato al Basic del vostro Commodore - 64.

Sono disponibili schedine porta EPROM di tipo diverso ed inoltre, possiamo fornire hardware con caratteristiche specificate da V/s dettagliata richiesta.

LA PRIMA SCHEDA INTELLIGENTE PORTA EPROM...

PER CBM-64 (*)



CREATE CON UNICARD

UNICARD accetta fino a tre EPROM (2764), un DIP-SWITCH permette di allocarle nella mappa di memoria del vostro computer ($8000, $A000, $C000, $E000) in ben 32 combinazioni diverse; inoltre UNICARD permette ben otto JMP e/o SYS automatici al sistema. Potete, finalmente, allocare i V/s programmi in C000... E000... senza caricarli dal disco o cassetta. Un tasto di RESET, con circuito di protezione ed un DIP-SWITCH permettono di utilizzare con profitto le V/s capacità di programmazione e di progettazione.

UNICARD CON MANUALE COSTA: L. 99.000

IVA COMPRESA!!

Cosmotron S.R.L.
ENGINEERING

00199 ROMA - Via A. Casella, 49 - Tel. (06) 8119406-8393950 - Tlx. 614593 TVRI

Per gli ordini inviare partita iva e/o codice fiscale. Merce in contrassegno, spese e spedizione a vs. carico.

Gruppo **CompuShop**

Sistemi per l'ufficio

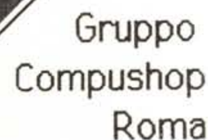
Vendita, consegna, installazione e assistenza personalizzata per:

Corsi di addestramento all'uso dei pacchetti applicativi

Rivenditore autorizzato programmi Italware

Concessionario 3M

Il nostro personale tecnico
e' a Vostra disposizione per dimostrazioni,
assistenza pre e post-vendita,
direttamente nel Vostro ufficio



Gruppo
Compushop
Roma
Via Nomentana
265/267/269/271/273
tel.(06) 857124-8450078