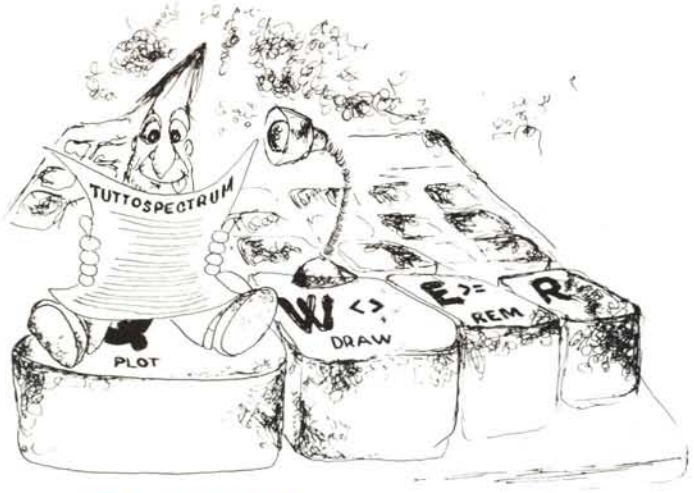


TUTTO SPECTRUM

di Fabio Schiattarella

Il linguaggio macchina sullo Spectrum

(prima parte)

Moltissimi tra voi, soddisfatti possessori di uno Spectrum, erano al momento dell'acquisto, ovvero quando ricevettero il computer in regalo, assolutamente digiuni "nell'arte della programmazione". Armati di pazienza, buona volontà, e spesso con l'unico ausilio del manuale di istruzioni accluso, avete acquisito la padronanza di questo strano linguaggio fatto di GOTO, IF... THEN, e così via. Capitolo dopo capitolo avete letto, appreso e digerito tutti i concetti fondamentali del Basic. A parecchi tuttavia sarà sicuramente rimasto oscuro il significato del capitolo 26, quello relativo al linguaggio macchina (ci riferiamo all'edizione originale inglese del manuale). Si tratta di appena tre pagine il cui argomento sembrerebbe a prima vista non avere stretta attinenza con il Basic. Il lettore che si accinge a scorrere le sue righe viene subito messo in guardia: "Attenzione — si dice all'inizio — questo capitolo è stato scritto per coloro che conoscono il linguaggio macchina dello Z80...". Ci piace immaginare, un po' prosaicamente forse, che il lettore di cui sopra, che da poco ha incominciato a introdursi ai "misteri della programmazione", debba sentirsi come il bambino che arrampicatosi in cima al recinto di un giardino incantato, scorge al di là un mondo nuovo e pieno di meraviglie, di cui però non può assaporare nulla. Anche il lettore già smaliziato, che qualcosa sa, non riesce a ricavare da quelle tre paginette un'idea chiara sulla possibilità di sfruttare il linguaggio macchina sullo Spectrum.

Sull'argomento esistono in commercio moltissimi libri. Alcuni trattano specificamente l'argomento "programmazione del microprocessore Z80" senza fare riferimento allo Spectrum. Si tratta in genere di libri piuttosto difficili e dedicati a persone già esperte di programmazione; parecchi contengono anche informazioni sull'hardware del microprocessore e sull'interfacciamento dello stesso con altri dispositivi. Appartiene a questa categoria il famoso testo di Zaks "Programmazione dello Z80". Inoltre questa stessa rivista sta pubblicando da diversi numeri (a partire dal numero 34 del mese di ottobre 1984) una serie di articoli riguardanti questo argomento.

Esistono poi moltissimi libri che trattano l'argomento facendo specifico riferimento allo Spectrum. Tra questi il lettore potrà trovare quello che si adatta meglio al suo grado di conoscenza della materia, sia egli un principiante o un esperto di programmazione in linguaggio macchina. Non è quindi nostra intenzione in questa sede iniziare un ennesimo corso sulla programmazione dello Z80 in linguaggio macchina; al contrario, a partire da questo numero vorremmo trattare approfonditamente le modalità di caricamento e interfacciamento dei programmi in linguaggio macchina nell'ambiente hardware/ software dello Spectrum. Cominceremo parlando dell'interfaccia tra ambiente Basic e linguaggio macchina; parleremo quindi delle modalità per caricare in memoria e lanciare un programma in l/m e dei metodi per il passaggio dei parametri tra ambiente Basic e l/m e viceversa. In seguito ci occuperemo della possibilità di utilizzare direttamente da l/m le varie routine contenute nella ROM, in modo da evitare di dover ritornare al Basic per eseguire le operazioni di input e output su cassetta, video, tastiera e stampante.

Assembler e linguaggio macchina

Conviene fare prima di tutto una distinzione tra linguaggio macchina (o codice macchina) e Assembler, due termini che spesso vengono erroneamente confusi. Il microprocessore Z80 ha un set di istruzioni di circa 400 elementi. Tali istruzioni vengono lette in memoria dove sono codificate sotto forma di byte. Per specificare una istruzione sono necessari da uno a quattro byte. Un byte è un numero binario lungo otto bit. Possiamo rappresentare questo numero mediante diverse notazioni, per esempio decimale o esadecimale. In particolare l'uso di una codifica in esadecimale costituisce una maniera molto comoda ed efficiente per maneggiare i byte. Poiché un carattere esadecimale rappresenta quattro bit, una coppia di caratteri esadecimali è sufficiente per rappresentare un byte. Si comprende quindi perché quando si rappresenta il contenuto della memoria si preferisca spesso usare la rappresentazione esadecimale piuttosto che quella decimale. In ogni caso un programma in linguaggio macchina altro non è se non una sequenza di numeri, compresi tra 00 e FF se si usa una codifica esadecimale, tra 0 e 255 se si utilizza una codifica decimale. Nello Spectrum, contrariamente ad altri computer, non è stato incorporato nel sistema operativo un monitor, ovvero un programma che accetti valori numerici e li carichi in memoria, permettendo poi di effettuare facili correzioni e la verifica sulla correttezza dei valori inseriti. Questo fatto obbliga il programmatore a caricare i byte utilizzando l'istruzione POKE che notoriamente accetta valori decimali compresi tra 0 e 255.

Il problema che si pone è quindi quello di maneggiare queste istruzioni in una forma comprensibile all'utente, in maniera tale che questo possa scrivere su carta i suoi programmi e possa poi caricarli in qualche modo nella memoria. La rappresentazione di istruzioni e quindi di programmi me-

diante sequenze di numeri ha il grande difetto di mancare di immediatezza. È evidente che una sequenza di numeri che rappresenta un programma dice assai poco su cosa un programma faccia effettivamente. Anche se fossimo dotati della memoria di Pico della Mirandola e ricordassimo a memoria le centinaia di codici esadecimali delle istruzioni, la lettura di un programma così codificato sarebbe un'operazione estremamente noiosa.

Questo ha fatto pensare alla possibilità di rappresentare le istruzioni non mediante valori numerici direttamente caricabili in memoria, ma mediante codici mnemonici che rappresentassero in qualche modo la funzione dell'istruzione. Consideriamo per esempio l'istruzione che somma al contenuto del registro A quello del registro B. Questa viene rappresentata mediante il valore esadecimale 80, corrispondente al decimale 128. L'idea è che se noi sostituissimo a questo singolo valore numerico l'espressione

```
SOMMA A, B
```

l'istruzione risulterebbe assai più comprensibile. Poiché lo Z80 è stato progettato negli Stati Uniti, i progettisti hanno assegnato a questa istruzione il nome simbolico di

```
ADD A, B
```

ovvero la parola inglese per somma. Questa rappresentazione non è in una forma direttamente eseguibile dal microprocessore, tuttavia ha il pregio di rendere subito l'idea di ciò che l'istruzione fa.

Nasce così il concetto di linguaggio assembly, in cui i valori numerici che rappresentano le istruzioni sono stati rimpiazzati da nomi simbolici.

L'operazione di traduzione tra codici simbolici mnemonici e istruzioni macchina direttamente caricabili può essere effettuata a mano, per esempio sfruttando la tabella riportata in fondo al manuale dello Spectrum a pagina 135. Si tratta chiaramente di un'operazione assai noiosa e viene di solito effettuata mediante appositi programmi detti assemblatori. Un programma assemblatore non si limita a questa opera di traduzione: consente, per esempio, di indicare indirizzi di memoria per mezzo di nomi simbolici definibili dall'utente.

Per valutare l'utilità di ciò consideriamo l'istruzione

```
CALL 0D6B
```

che provoca la chiamata della routine della ROM che cancella lo schermo. L'assemblatore permette di associare al valore 0D6B un nome simbolico, per esempio CANCELLA, e quindi di scrivere

```
CALL CANCELLA
```

il programma ne guadagna senz'altro in leggibilità.

Non si tratta quindi di variabili, concetto praticamente assente nella programmazione in linguaggi a basso livello come l'Assembler, ma semplicemente della possibilità di indicare valori numerici costanti per mezzo di nomi. Oltre ad assemblare il programma, l'assemblatore mette a disposizione dell'utente molte altre "facilities", come un editor per scrivere o fare correzioni, la verifica della eventuale presenza di errori sintattici nel programma, la possibilità di salvare su cassetta o microdrive programmi già scritti, ovvero di listarli su stampante. Si tratta di un insieme minimo di funzionalità comuni a tutti i programmi assemblatori in commercio. Per lo Spectrum ne sono disponibili sul mercato diversi, e quasi tutti consentono di eseguire molte più operazioni di quante noi ne abbiamo descritte. Vogliamo anche ricordare che insieme all'assemblatore viene quasi sempre venduto un secondo programma, detto monitor, che risulta estremamente utile nella fase di debugging dei programmi in linguaggio macchina, ovvero nella fase di ricerca di errori e messa a punto. Un monitor permette di visualizzare o alterare in ogni istante il contenuto dei registri della CPU e di un'area qualsiasi della memoria, permette anche di eseguire il programma in modalità passo passo, permettendo nel contempo all'utente di vedere come si modifica il contenuto dei registri e della memoria. È anche possibile fare il passaggio inverso da linguaggio macchina a linguaggio assemblativo, ovvero di disassemblare un pezzo di codice macchina. Anche in questo caso si tratta di un insieme minimo di funzionalità e in generale i programmi commerciali possiedono molte più possibilità di quante noi ne abbiamo qui descritte.

Un assemblatore e un monitor sono due programmi che non dovrebbero assolutamente mancare a chi abbia l'intenzione di dedicarsi assiduamente allo studio e all'applicazione del linguaggio macchina sullo Spectrum.

```
ROUTINE MEMORIA LIBERA
5B00              10         ORG 23296 ; CARICA IL PROGRAMMA NEL
                                         BUFFER  STAMPANTE

5C65              20 STKEND  EQU 23635
5B00  210000      30         LD HL,00  ; ARTIFICIO PER COPIARE IN
5B03  39          40         ADD HL,SP ; HL IL VALORE DELLO
                                         ; STACK POINTER
5B04  ED5B655C    50         LD DE,(STKEND)
5B08  A7          60         AND A     ; SERVE AD AZZERARE IL
                                         CARRY FLAG
5B09  ED52        70         SBC HL,DE ; ESEGUE LA SOTTRAZIONE
                                         TRA I VALORI DI INIZIO
                                         E FINE DELLA MEMORIA
                                         BASIC
5B0B  44          80         LD B,H    ; TRASFERISCE IN BC IL
5B0C  4D          90         LD C,L    ; RISULTATO CONTENUTO
                                         IN HL
5B0D  C9          100        RET


 10 REM LA ROUTINE E' ALLOCATA NEL BUFFER DELLA STAMPANTE
100 FOR I=23296 TO 23309
110 READ B
120 POKE I,B
130 NEXT I
140 DATA 33,0,0,57,237,91,101,92,167,237,82,68,77,201
150 REM PER SAPERE LA MEMORIA DISPONIBILE PER IL BASIC USARE
PRINT USR 23296
```

Figura 1

Pro e contro

Come forse già saprete il vantaggio principale del linguaggio macchina per l'utente di piccole macchine, come lo Spectrum, consiste nella sua velocità. Non è possibile fare un confronto immediato con il Basic in quanto l'incremento di velocità dipende dal tipo di operazione, tuttavia per quasi tutte le funzioni il linguaggio macchina risulta essere enormemente più veloce del Basic interpretato. Questo incremento di velocità risulta assai comodo dovendo realizzare programmi molto complessi e contenenti cicli che vengono eseguiti un grande numero di volte; diventa poi essenziale nelle applicazioni in tempo reale, per esempio, dovendo realizzare delle animazioni o il movimento di figure sullo schermo.

Un secondo vantaggio consiste nel fatto che, utilizzando il linguaggio macchina, si ha una visione più chiara ed un accesso completo alle risorse della macchina. Supponiamo per esempio di utilizzare delle variabili booleane, ovvero variabili che possono assumere solo i due valori vero o falso. In Basic, dove tale possibilità non è prevista, possiamo utilizzare una variabile numerica, per esempio BOOL, cui assegnare i valori 1 e 0 per vero e falso. Ogni variabile numerica occupa in memoria cinque byte, più lo spazio necessario per memorizzarne il nome ogni volta che questo appare nel listato. Tutto ciò ci porta a consumare parecchi byte di memoria laddove un solo bit sarebbe stato sufficiente. In linguaggio macchina, al contrario, abbiamo un insieme di istruzioni per leggere e scrivere nei singoli bit dei registri della memoria.

Tra i vantaggi occorre considerare an-

che il punto di vista didattico. Lo studio dell'hardware dello Spectrum e del suo sistema operativo ci consente di acquisire maggiori conoscenze sul funzionamento dei microprocessori e dei sistemi a microcomputer, conoscenze che non si limitano al solo funzionamento dello Spectrum, ma che una volta acquisite sono facilmente trasportabili anche ad altri sistemi.

Dopo avere accennato ad alcuni dei pregi del linguaggio macchina occorre naturalmente presentare l'altra faccia della medaglia. Il difetto principale del linguaggio macchina consiste nella scarsa potenza delle sue istruzioni. Se da una parte questo conviene perché, come abbiamo visto nell'esempio sopra, ci consente un accesso più immediato e capillare alle risorse, dall'altra ci porta ad avere programmi molto complicati e lunghi anche per svolgere compiti molto semplici e che in Basic richiederebbero pochissime istruzioni. Per esempio l'esecuzione di una moltiplicazioni in Basic non comporta nessuna difficoltà, mentre l'implementazione di una moltiplicazione in linguaggio macchina richiede parecchie decine di istruzioni. Nel linguaggio macchina, inoltre, non sono presenti istruzioni immediate per l'input da tastiera, ovvero per l'output su video e stampante, così come mancano istruzioni per il salvataggio e il caricamento dei programmi da cassetta o microdrive. Per fare queste operazioni è necessario appoggiarsi alle routine che sono presenti nella ROM, oppure scriverne delle proprie.

Un programma in linguaggio macchina che svolga una funzione qualsiasi, senza essere interfacciato con il resto del sistema, non è in grado di fare granché perché non è in grado di ricevere informazioni ovvero fornire dati al mondo esterno.

Caricamento

Di tutto ciò parleremo più avanti. Per ora iniziamo a vedere una cosa banale, se si vuole, ma fondamentale: come si carica in memoria un programma in linguaggio macchina e come lo si fa partire.

Il primo problema che possiamo porci è in quale zona della memoria andare a mettere il nostro programma in linguaggio macchina. Sono possibili diverse scelte. Il modo generalmente più conveniente per allocare un programma è memorizzarlo nella parte alta della memoria, riservandogli un'area apposita. Il Basic dello Spectrum prevede espressamente questa possibilità per mezzo dell'istruzione CLEAR. Il formato di questa istruzione è

```
CLEAR indirizzo
```

l'effetto è di limitare l'area disponibile per i programmi Basic fino all'indirizzo specificato. Tutto lo spazio a partire dall'indirizzo successivo, fino alla fine della memoria (esclusi eventualmente gli ultimi 128 byte in cui sono memorizzati i caratteri definibili dall'utente) rimane disponibile per i nostri programmi in linguaggio macchina. In questo modo il programma in linguaggio macchina non può essere cancellato mediante l'istruzione NEW, che ha come effetto quello di cancellare solo l'area riservata al Basic. Ciò può risultare molto utile in quanto protegge il programmatore da eventuali cancellazioni accidentali e in quanto permette di eliminare un programma Basic divenuto inutile, come un caricatore, senza perdere il codice macchina. Chiaramente quando si va a limitare l'area riservata al Basic bisogna porre attenzione a non ridurla troppo, ma a dimensionarla secondo la grandezza del programma Basic stesso e del numero di variabili che esso utilizza.

Un metodo che può risultare molto utile, se il programma da caricare non è più lungo di 256 byte e non si deve utilizzare la stampante, è quello di allocare il programma nel buffer stampante ovvero nelle 256 locazioni di memoria successive all'indirizzo 23296.

È anche possibile caricare un programma in linguaggio macchina nel corpo stesso di un programma Basic all'interno di una linea di REM. La sequenza delle operazioni da eseguire è questa. Come prima istruzione del programma Basic si mette una REM seguita da tanti caratteri quanti sono i byte del programma in linguaggio macchina da caricare. Tali caratteri possono essere di tipo qualsiasi, quello che conta è il loro numero. Se si dispone di uno Spectrum in versione base, senza Interface 1 e microdrive, l'indirizzo iniziale del Basic si trova sempre alla stessa posizione, ovvero all'indirizzo specificato nella coppia di variabili di sistema PROG. Per conoscere tale valore basta calcolare l'espressione

```
PRINT PEEK 23635 + (256*PEEK 23636).
```

Poiché il numero della linea occupa due byte e altrettanti ne richiede la lunghezza della stessa, aggiungendo il byte necessario per il codice della REM abbiamo che cinque byte dopo l'inizio dell'area destinata al programma Basic possiamo cominciare a caricare il programma in linguaggio macchina. Ripetiamo che è assolutamente necessario che l'istruzione REM che contiene i codici sia la prima del programma. Il vantaggio di questo sistema è di poter salvare il programma in linguaggio macchina unitamente al programma Basic. Si tratta di un metodo ben conosciuto dai possessori dello ZX81, in cui non è possibile caricare o salvare su cassetta dati di tipo CODE, e quindi si rende necessario ricorrere a questo espediente per poter salvare programmi in linguaggio macchina. Non avendo lo Spectrum tale limitazione ci sembra che questo metodo sia una inutile complicazione.

A questo punto, dopo avere deciso la zona di memoria in cui andrà posto il programma, occorrerà caricare i codici all'interno della stessa. A questo scopo potete usare un monitor decimale o esadecimale, oppure un assemblatore.

Se prevedete che il vostro programma debba rientrare al Basic è necessario porre al punto del rientro una istruzione RET il cui codice esadecimale è C9 (decimale 201).

Una volta caricato il programma in memoria il nostro consiglio più caldo è quello di salvarlo immediatamente, prima di cercare di farlo girare. L'esperienza ci dice che quasi nessun programma funziona correttamente la prima volta che viene fatto girare. Ciò risulta particolarmente vero per i programmi in linguaggio macchina, dove sbagliare è assai più semplice. Tuttavia mentre in Basic una istruzione non corretta può provocare al più una segnalazione d'errore, in linguaggio macchina le conseguenze di un errore sono sempre imprevedibili e conducono assai spesso ad un blocco del sistema o ad un reset generale, con la conseguente perdita del programma appena caricato.

Esecuzione

Dopo avere salvato il nostro programma occorre quindi farlo girare, di questo si occupa la funzione USR. Il formato è

```
istruzione USR indirizzo
```

dove indirizzo è l'indirizzo iniziale del programma in linguaggio macchina che vogliamo lanciare, espresso mediante un valore decimale compreso tra 0 e 65535. Istruzione può essere una qualsiasi istruzione che accetti un argomento numerico. La funzione USR produce come effetto quello di lanciare un programma in linguaggio macchina e restituisce come valore il contenuto della coppia di registri BC espresso mediante un numero decimale compreso tra 0 e 65535. Facciamo subito un esempio.

Consideriamo il semplicissimo programma

```
LD BC, 99
RET
```

che ha come unico scopo quello di caricare la coppia di registri BC con il valore decimale 99. La corrispondenza tra nomi simbolici e codici è la seguente

```
LD BC, numero        1
RET                  201
```

quindi la codifica decimale del programma è

```
1 99 0 201
```

che possiamo caricare in memoria mediante un semplice programma Basic a partire dall'indirizzo 32000.

Notate come sia stato necessario per codificare 99, utilizzare una coppia di numeri (0 e 99). Questo perché i registri da caricare sono due, B e C.

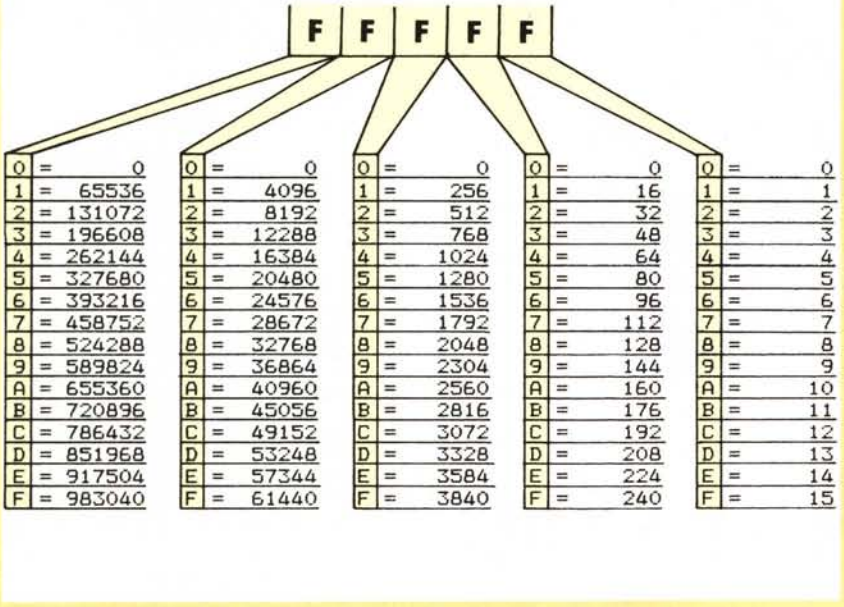
```
10 CLEAR 31999
20 FOR I=32000 TO 32003
30 READ D
40 POKE I,D
50 NEXT I
60 DATA 1,99,0,201
```

a questo punto possiamo digitare come comando

```
PRINT USR 32000
```

e vedremo apparire sul video il numero 99.

Cosa è successo? La chiamata della funzione USR ha avuto come effetto quello di lanciare il programma posto a partire dalla locazione 32000. L'esecuzione di questo è consistita nel caricare nella coppia BC il valore 99, e l'istruzione RET ha prodotto il



F	F	F	F	F

0 = 0	0 = 0	0 = 0	0 = 0	0 = 0
1 = 65536	1 = 4096	1 = 256	1 = 16	1 = 1
2 = 131072	2 = 8192	2 = 512	2 = 32	2 = 2
3 = 196608	3 = 12288	3 = 768	3 = 48	3 = 3
4 = 262144	4 = 16384	4 = 1024	4 = 64	4 = 4
5 = 327680	5 = 20480	5 = 1280	5 = 80	5 = 5
6 = 393216	6 = 24576	6 = 1536	6 = 96	6 = 6
7 = 458752	7 = 28672	7 = 1792	7 = 112	7 = 7
8 = 524288	8 = 32768	8 = 2048	8 = 128	8 = 8
9 = 589824	9 = 36864	9 = 2304	9 = 144	9 = 9
A = 655360	A = 40960	A = 2560	A = 160	A = 10
B = 720896	B = 45056	B = 2816	B = 176	B = 11
C = 786432	C = 49152	C = 3072	C = 192	C = 12
D = 851968	D = 53248	D = 3328	D = 208	D = 13
E = 917504	E = 57344	E = 3584	E = 224	E = 14
F = 983040	F = 61440	F = 3840	F = 240	F = 15

Tabella di conversione esadecimale decimale

L'uso di questa tabella (già pubblicata nel numero 19 di MC) permette di convertire a mano numeri da esadecimali a decimali e viceversa, fino a un valore di 1.028.575 in decimale (FFFFF in esadecimale). Questo range di valori è più che sufficiente per il tipo di applicazioni di cui ci interessiamo.

Dovendo convertire un numero intero da esadecimale a decimale la procedura è la seguente. Si allinea il numero a destra in corrispondenza della griglia di riferimento nella figura (dove sono state riportate le cinque F). Per ciascuna posizione si prende nella relativa colonna, il numero decimale corrispondente alla cifra esadecimale del numero da convertire. Se un numero non occupa tutte le posizioni della griglia, si assume che ci sia zero nelle posizioni lasciate libere. Infine si fa la somma dei valori decimali prelevati dalle tabelle. Facciamo un esempio: dato il numero 5EA7, preleviamo

5 → 20480 dalla seconda colonna
E → 3584 dalla terza colonna
A → 160 dalla quarta colonna
7 → 7 dall'ultima colonna

20480 + 3584 + 160 + 7 = 24231

dalla prima colonna non abbiamo prelevato alcun valore perché il numero era composto di sole quattro cifre esadecimali.

Dovendo convertire un numero da decimale a esadecimale la sequenza di operazioni è un po' più complessa. Cerchiamo in tutte le colonne il più grande valore nella tabella che sia più piccolo o uguale al numero da convertire; ossia un valore tale che non ne esista nella tabella un altro compreso tra questo e il numero da convertire. Il valore esadecimale corrispondente costituisce la cifra più significativa (quella più a sinistra) del numero esadecimale da trovare. Facciamo la sottrazione tra il numero di partenza e il suo minore trovato nella tabella. Con la differenza tra i due valori ripetiamo da capo il procedimento illustrato, andando a cercare il minore nella colonna immediatamente a destra di quella in cui abbiamo prelevato la prima cifra. Porremo la corrispondente cifra esadecimale a destra della parte già calcolata. Nel caso in cui tutti i valori decimali di una colonna, tranne lo zero, risultassero maggiori del resto del numero da convertire, metteremo zero e proseguiremo con la colonna successiva.

Si ripete il procedimento fino all'ultima colonna.

Consideriamo per esempio il numero 53440.

1) Il numero immediatamente inferiore nella tabella è 53248.
2) 53248 → D (la cifra più significativa del numero)
3) 53440—53248 = 192
4) Ripetiamo il procedimento con 192 a partire dalla terza colonna.
5) 192 è più piccolo di tutti i numeri della terza colonna tranne 0
6) 192 → 0
7) Passiamo alla seconda colonna
8) 192 è presente nella tabella
9) 192 → C
10) Il resto è 0 quindi poniamo uno 0 per la quarta colonna. Il risultato quindi è D0C0

ritorno al Basic. A questo punto la funzione USR ha restituito il valore contenuto nella coppia BC al momento di rientrare al Basic e cioè proprio 99. La print infine, che aveva a sua volta chiamato la USR, ha provveduto a stampare tale valore.

La USR può essere quindi inserita in un qualsiasi contesto in cui sia consentito avere un valore numerico. In particolare possiamo associarla direttamente a parecchi comandi Basic ottenendo, oltre al lancio del programma in linguaggio macchina, altri utili effetti collaterali. Analizziamo i casi più interessanti.

PRINT USR: lancia il programma in linguaggio macchina e stampa il contenuto di BC al momento di rientrare al Basic.

LET <var> = USR: lancia il programma in l/m e registra nella variabile <var> il valore finale di BC.

RUN USR: esegue il programma in linguaggio macchina e alla fine lancia il programma Basic cominciando dalla linea il cui numero è maggiore o uguale del contenuto di BC.

GOTO USR: ha lo stesso effetto della RUN USR, ma al contrario di questa non azzera le variabili prima di lanciare il programma Basic.

RAND USR: è la forma più usata in quanto apparentemente priva di "effetti collaterali", dopo avere lanciato il programma in linguaggio macchina, con il contenuto di BC aggiorna il seme del generatore di numeri casuali.

Facciamo un ulteriore esempio. Il programmino di figura 1 calcola la quantità di memoria libera disponibile per i programmi Basic, misurando lo spazio compreso tra la cima dell'area contenente il programma Basic, il cui indirizzo è contenuto nella variabile di sistema STKEND, e l'inizio della zona variabili, il cui indirizzo coincide con la cima dello stack del microprocessore Z80 ed è quindi contenuto nel registro SP del microprocessore stesso.

Il programma viene lanciato mediante una PRINT USR secondo le modalità sopra illustrate.

Abbiamo visto come sia possibile in maniera semplice ed efficiente passare almeno un valore da linguaggio macchina al programma chiamante. Se i valori sono più di uno questa tecnica si rivela già inefficace.

Nella prossima puntata vedremo come sia possibile risolvere tale problema e soprattutto il problema inverso, ovvero quello del passaggio di valori da un programma Basic ad uno in linguaggio macchina. I progettisti della Sinclair, infatti, non hanno previsto nessuna opportunità per svolgere tale operazione. Vedremo come è possibile eliminare questa restrizione in modo semplice ed efficiente, ma soprattutto in maniera completamente trasparente all'utente. MC