i trucchi del CP/M

di Pierluigi Panunzi

MBASIC

Siamo ormai lanciatissimi nell'intento di creare nuove istruzioni dell'MBASIC, sempre per quanto riguarda la versione 5.21.

A proposito di questo, invitiamo i lettori, oltre che a contribuire con spunti originali, a segnalare anche eventuali adattamenti ad altre release *dell'MBASIC di tutte le nuove istruzioni che via via presentiamo.*

Inoltre consigliamo ancora una volta e vivamente di rileggersi la prima puntata di questa rubrica, in quanto tutte le nuove istruzioni si possono adattare solo **dopo** *aver effettuato le opportune modifiche all'interprete stesso: infatti senza tali modifiche le nuove istruzioni non funzionerebbero assolutamente.*

Viceversa è molto utile avere sottomano la seconda puntata in quanto utilizzeremo delle informazioni, che sarebbe inutile ripetere, ma che sono determinanti per la comprensione di alcuni concetti che esprimeremo.

In questa puntata presenteremo dunque due istruzioni veramente utili, specialmente a coloro che seguono in particolare rubriche come questa....

Abbandoniamo le ormai obsolete **PEEK** *e* **POKE***!!!*

Impariamo invece ad usare le nuovissime **RDA** *e* **STO***!!!*

Miglioriamo la vecchia ma gloriosa PEEK

Dell'istruzione **PEEK** (e della sua parente **POKE**) non se ne può dire che del bene: quanti "smanettomani" hanno cominciato la loro opera devastatrice del proprio povero personal computer grazie a questa semplicissima istruzione?!

E quanti ancora, non contenti delle possibilità del suddetto personal non hanno cominciato a punzecchiarlo qua e là. POKando i valori più inverosimili nelle celle di memoria meno opportune, pur di avere la soddisfazione di aver così inventato qualcosa di nuovo, a cominciare dal "crash" del sistema operativo?!...

Un ostacolo che si incontra indubbiamente non appena ci si addentra nel mondo del linguaggio macchina (lanciandosi così dal "trampolino"-Basic) è la necessità di lavorare con valori esadecimali a 16 bit: mentre infatti la **PEEK** estrae il contenuto di un byte di memoria, per avere il contenuto di due byte consecutivi di memoria (in generale un **indirizzo**) bisogna usare un' espressione, che in genere si confina in un **DEF FNx.**

Se la variabile **A** contiene un indirizzo di memoria, per leggere cosa c'è in **A,A+1**, bisogna ad esempio impostare

```
PRINT PEEK (A) + 256*PEEK(A+1)
```

in quanto correntemente un valore a 16 bit è posto in memoria in configurazione "Low/High" e cioè prima il byte meno significativo (LSB) e poi quello più significativo (MSB).

Ecco che perciò appare chiaro cosa si debba pretendere da una nuova istruzione: l'estrazione di un valore a 16 bit. L'istruzione **RDA** (che starebbe per "ReaD Address") è quanto fa per noi: ritornando all'esempio precedente, ora si può impostare più semplicemente

```
PRINT RDA (A)
```

Considerato che poi l'implementazione di tale funzione in linguaggio macchina richiede appena 13 byte, ci si può domandare perché non ci abbiano pensato prima...

E pensare che già la **PEEK** consta di 10 byte!!

Comunque passiamo alla modifica da apportare al nostro Basic **già modificato**, che avevamo ribattezzato con **PBASIC.COM**.

Eccoci dunque pronti: dal CP/M impostiamo

```
A>ZSID PBASIC.COM
```

oppure, al solito,

```
A>DDT PBASIC.COM
```

ed effettuiamo le modifiche che ora riportiamo.

Di seguito ne vedremo il significato.

Le modifiche sono:

```
a) 608E  ..  CD
   608F  ..  CC
   6090  ..  22      CALL 22CCH
   6091  ..  CD
   6092  ..  41
   6093  ..  5D      CALL 5D41H
   6094  ..  7E      LD A, (HL)
   6095  ..  23      INC HL
   6096  ..  66      LD H, (HL)
   6097  ..  6F      LD L,A
   6098  ..  C3
   6099  ..  CD
   609A  ..  29      JP 29CDH
b) 01DF  00  8E
   01E0  00  60      l'indirizzo di partenza
c) 03E8  4D  CD      la lettera M di "RANDOM"
   03E9  49  BB      token di "RANDOM"
   03EA  5A  44      la lettera D di "RDA"
   03EB  C5  C1      la lettera A di "RDA"
   03EC  BB  21      il token di "RDA"
```

Partiamo, per la spiegazione, dal fondo e cioè dal punto c).

La scorsa puntata abbiamo parlato di quella zona di memoria dove sono presenti i nomi delle istruzioni MBASIC: abbiamo detto che tali nomi sono tutti troncati (per economia!) dell'iniziale, hanno l'ultima lettera con il bit più significativo posto ad 1 e sono seguite dal rispettivo "token", con il quale vengono effettivamente memorizzate in un programma Basic.

Avevamo pure visto che per creare nuovi "nomi" di istruzioni avevamo bisogno di "istruzioni-dal-nome-lungo".

Abbiamo scelto questa volta la chilometrica **RANDOMIZE** da adesso in poi chiamata **RANDOM**, per far spazio alla più corta RDA (vi ricordate la questione dell'iniziale?).

Procedendo a ritroso, nel punto b) andiamo a scrivere nella tabella degli indirizzi delle varie istruzioni proprio quello di partenza della nostra routine: a questo proposito, se si desiderasse porre tale routine in altra zona di memoria (data la sua completa rilocabilità) basterà cambiare il valore dell'indirizzo iniziale nel punto b).

Ed infine nel punto a) la nostra "grande" routine:

— le due subroutine **22CCH** e **5D41H** sono due routine di sistema che effettuano la scansione del testo Basic in cerca di un'espressione intera (quindi sia un valore decimale o esadecimale, sia il nome di una variabile) ed in quest'ultimo caso vanno ad estrarne il valore dalla memoria, in modo tale da avere in **HL** proprio l'indirizzo della coppia di byte da esaminare.

Le istruzioni successive non servono altro che a caricare in **HL** stesso il **contenuto** delle celle in questione: tra l'altro il metodo usato è quello solito, che sfrutta le possibilità di indirizzamento indiretto proprie dello Z80.

L'ultimo salto invece invia alla routine chiamante il contenuto di **HL**, per poterne consentire l'elaborazione.

Va da sè che la nuova istruzione si può porre in statement del tipo

```
X=RDA (&H3333)
```

oppure

```
IF RDA (I)>10000 THEN....
```

insomma come una qualsiasi altra istruzione.

Ora miglioriamo la POKE

Perché non inventarsi qualcosa di simile per quanto riguarda la **POKE**?

Noi abbiamo fatto ancora di meglio!!!

Supponiamo che da programma dobbiamo impostare una routine in linguaggio macchina in una certa zona di memoria: dobbiamo perciò (finora!) armarci delle interminabili linee di **"DATA"**, di solito foriere di errori, nonché di un loop di lettura e scrittura in memoria (leggasi **READ** e **POKE** all'interno di un ciclo **FOR... NEXT**).

Cioè

```
10 DATA 1,2,3,4,5,6,7,8,9,0
20 FOR I=1 TO 10
30 READ A
40 POKE ADDR, A
50 ADDR=ADDR+1
60 NEXT
```

dove ADDR è l'indirizzo iniziale della zona di memoria interessata e dove abbiamo espanso in 6 linee (per chiarezza) ciò che si poteva scrivere in due-tre linee.

Sarebbe molto comodo poter scrivere un'istruzione del tipo

```
10 STO ADDR, 1,2,3,4,5,6,7,8,9,0
```

O no?!

Ebbene, eccola qui!!! Pronta per l'uso...

Ancora una volta si è ottenuto un notevolissimo risultato con il minimo sforzo.

Pensate: la routine che implementa il tutto è ancora una volta brevissima, appena 22 byte!

Vediamo dunque le modifiche:

```
a)  603F  ..  CD
    6040  ..  C2
    6041  ..  22   CALL 22C2H
    6042  ..  D5   PUSH DE
    6043  ..  CD
    6044  ..  41
    6045  ..  5D   CALL 5D41H
    6046  ..  CD
    6047  ..  5C
    6048  ..  20   CALL 205CH
    6049  ..  D1   POP DE
    604A  ..  12   LD (DE) ,A
    604B  ..  13   INC DE
    604C  ..  D5   PUSH DE
    604D  ..  2B   DEC HL
    604E  ..  CD
    604F  ..  05
    6050  ..  13   CALL 1305H
    6051  ..  20
    6052  ..  F3   JR NZ, F3
    6053  ..  D1   POP DE
    6054  ..  C9   RET
b)  013D  C9  3F
    013E  OC  60   l'indirizzo iniz...
c)  041C  53  D3   S di "SYS"
    041D  54  BD   token di "SYS"
    041E  45  54   T di "STO"
    041F  CD CFO   di "STO"
    0420  BD  9C   token di "STO"
```

Analogo discorso andrebbe fatto per la nuova istruzione, ma non lo facciamo se non per dire che la vecchia **SYSTEM** d'ora in poi si chiamerà **SYS** (attenzione! Non ha nulla a che vedere con l'omonima di Commodoriana memoria!)

Per quanto riguarda la routine vera e propria, c'è da dire che anche questa è completamente rilocabile ed in particolare le varie istruzioni hanno i seguenti significati:

— le prime due chiamate a subroutine sono praticamente le stesse di quelle riguardanti la **RDA**: in particolare in **DE** avremo l'indirizzo iniziale della zona di memoria considerata;

— la **205CH** è stata scovata all'interno di un'altra routine di sistema e cerca se nel testo vi sono espressioni intere separate da una virgola (è proprio quello che ci voleva!);

— la **1305H** invece avanza nel testo e se incontra un "blank" o un carattere ":" ritorna con il flag Z posto ad "1";

— se si trova invece qualcos'altro si torna indietro nel loop.

Va da sè che la routine (come d'altro canto la precedente) è completamente protetta da errori di sintassi e di altro tipo.

È inutile perciò inserire in una locazione di memoria una stringa, invece di un valore esadecimale...

Con ciò abbiamo terminato e diamo l'appuntamento ai lettori alla prossima puntata. MC

Elenco del software disponibile su cassetta o minifloppy

Per ovviare alle difficoltà incontrate da molti lettori nella digitazione dei listati pubblicati nelle varie rubriche di software sulla rivista, MCmicrocomputer mette a disposizione i programmi più significativi direttamente su supporto magnetico. Riepiloghiamo qui a fianco i programmi disponibili per le varie macchine, ricordando che i titoli non sono previsti per computer diversi da quelli indicati.
Il numero della rivista su cui viene descritto ciascun programma è riportato nell'apposita colonna; consigliamo gli interessati di procurarsi i relativi numeri arretrati, eventualmente rivolgendosi al nostro Servizio Arretrati utilizzando il tagliando pubblicato in fondo alla rivista.

Per l'ordinazione inviare l'importo (a mezzo assegno, c/c o vaglia postale) alla Technimedia srl, Via Valsolda 135, 00141 Roma.

Le cassette utilizzate sono Basf C-60 Compusette II; i minifloppy sono Basf singola faccia singola densità.

Codice	Titolo programma	MC n.	Prezzo	Note
	APPLE II			
DA2/00	Shape Tablet	22	15000	
DA2/01	Motomuro	26	15000	
DA2/02	&DEBUG	28	15000	
DA2/03	EDIT + INPUT	29	15000	
DA2/04	Basic modulare	34	15000	
DA2/05	ANNA Animation Lang.	35	15000	
	COMMODORE 64			
C64/01	Briscola	25	17000	
C64/02	Serpentone	29	17000	
C64/03	Othello	29	17000	
C64/04	Chase	33	17000	
C64/05	Spreadsheet	34	30000	
C64/06	Bilancio familiare	35	17000	
C64/07	The dark wood	36	17000	
D64/01	Spreadsheet	34	15000	
	COMMODORE VIC-20			
CVC/01	VIC-Maze	19	17000	Config. base
CVC/02	Pic-Man	23	17000	Config. base
CVC/03	Briscola	25	17000	Config. base
CVC/04	Grand Prix	28	17000	Config. base
CVC/05	Frogger	26	17000	RAM: almeno + 3 K
CVC/06	Invaders	29	23000	RAM: + 16 K
CVC/07	Othello	29	17000	RAM: + 16 K
CVC/08	SKI	31	17000	Config. base
CVC/09	VIC-quiz	32	17000	RAM: almeno + 8 K
CVC/10	Zigurat	33	17000	Config. base
CVC/11	Extended Basic	36	17000	RAM: + 16 K
CVC/12	Fireman	36	17000	Config. base
DVC/01	EXMA	27/28	15000	RAM: + 16 K
	SINCLAIR SPECTRUM			
CSS/01	TRILAB	28	17000	
CSS/02	SET di caratteri	27/29	17000	
CSS/03	Grafica TREDIM	29	17000	
CSS/04	Ippica	30	17000	
CSS/05	Graphic-Comp	32	17000	
CSS/06	Macchina del tempo	34	17000	
CSS/07	Piramide di Iunnuh	35	17000	
	TEXAS TI-99/4A			
CT9/01	Macchina del tempo	27	17000	
CT9/02	Simon	29	17000	
CT9/03	Babilonia	30	17000	
CT9/04	Labirinto 3D	31	17000	
CT9/05	Piramide di Iunnuh	33	17000	Extended Basic
CT9/06	Scrabble	34	17000	
CT9/07	Morphy	35	17000	

Nota:
l'iniziale del codice e' C per le cassette, D per i minifloppy