A cura di Pierluigi Panunzi

Calcolo di espressioni (3ª parte)

Concludiamo con questa puntata l'analisi del calcolo di un'espressione logico-algebrica, dal punto di vista dell'interprete MBASIC, parlando di due routine: la prima è l'ormai omnipresente «43C7H» incontrata praticamente in ogni puntata. La seconda invece è una subroutine usata moltissime volte dalle funzioni dell'MBASIC e che perciò incontreremo già da questa puntata: è nostra intenzione infatti andare ad analizzare piano piano tutte le funzioni ed i comandi dell'MBASIC, senza però disdegnare eventuali «salti all'indietro» per proporre argomenti da affrontare prima di proseguire nell'analisi.

La subroutine 43C7H

Abbiamo parlato ormai moltissime volte di questa subroutine, che rappresenta in un certo senso un'eccezione nell'ambito delle svariate subroutine utilizzate dall'interprete sotto analisi.

La differenza con le altre subroutine viene evidenziata dalla considerazione che tale subroutine, nel suo funzionamento, effettua la lettura del testo del programma MBASIC alla ricerca di un particolare byte, che «deve» essere trovato, pena la segnalazione di errore con conseguente arresto del programma stesso.

Il fatto importante è che il byte «da trovare» è posto nel byte successivo alla chiamata, potendo assumere teoricamente qualsiasi valore esadecimale compreso tra ØØH e ØFFH: ciò comporta che andando a disassemblare una certa routine, subito dopo la chiamata alla subroutine 43C7H, non dobbiamo disassemblare il byte successivo, ma dobbiamo indicarlo genericamente con una direttiva «DEFB», cosa che abbiamo fatto tutte le volte...

Non rispettando questo accorgimento ci si troverebbe spaesati se non fuori strada, dal momento che opportuni valori del «byte» possono venir interpretati come istruzioni, peggio se a più byte.

Ricordiamo perciò la chiamata completa a tale subroutine, supponendo di voler trovare nel byte successivo di testo una parentesi aperta, espressa come codice ASCII dal valore esadecimale 28H: in tal caso la chiamata sarebbe la seguente

```
CALL 43C7H; scan del testo
DEFB 28H   ; "(" espressa in ASCII
```

usata soprattutto dalle funzioni dotate di argomento da porre tra parentesi, appunto per verificare la presenza della parentesi aperta all'inizio, subito dopo il «token» relativo alla funzione in esame.

Facendo riferimento al listato 1, vediamo che la subroutine 43C7H è abbastanza corta, mentre in realtà alla fine, proprio le ultime due istruzioni, si riaggancia ad altrettante routine delle quali una conosciamo già: ma andiamo con ordine.

Per comprendere meglio il funzionamento di tale subroutine, ripetiamo unica nel suo genere all'interno dell'MBASIC (peccato...), ricordiamo brevemente cosa succede all'atto della chiamata ad una subroutine: sappiamo che in tal caso viene salvato nello stack il cosiddetto «indirizzo di ritorno» della subroutine stessa e cioè il punto al quale si ritornerà per effetto della RET posta al termine della subroutine in esame.

```
43C7H    LD A,(HL)
         EX (SP),HL
         CP (HL)
         JP NZ,43D7H
         INC HL
         EX (SP),HL
         INC HL
         LD A,(HL)
         CP 3AH
         RET NC
         JP 130AH
43D7H    JP 0CC9H
```

Listato 1 - Routine di ricerca di un dato byte all'interno del testo di un programma MBASIC.

Osservando meglio le istruzioni che implementano la chiamata, notiamo che l'indirizzo posto sullo stack è proprio quello in cui è posto il «byte» da trovare, come dire che l'indirizzo posto nello stack è proprio il «puntatore» al byte desiderato.

Analizzando dunque la routine del listato 1, osserviamo che in accumulatore viene posto il byte da analizzare della linea di programma MBASIC (il cui indirizzo era come al solito posto in HL).

Subito dopo viene usata un'istruzione particolarissima dell'8080 nonché dello Z80, in generale non molto usata se non in situazioni particolari come questa: tale istruzione (è la **«EX (SP), HL»** per lo Z80 oppure la **«SPHL»** per l'8080) effettua lo scambio bidirezionale tra i contenuti della coppia di registri HL e del «top dello stack». Per effetto di tale istruzione perciò ora HL punterà al byte definito con una «DEFB» (ricordate che era l'indirizzo di ritorno della subroutine appena chiamata?!), mentre ora nello stack avremo «salvato» il vecchio valore contenuto di HL e cioè il puntatore al testo del programma.

Detto ciò, dunque, la successiva **«CP (HL)»** effettua la comparazione tra il contenuto dell'accumulatore (il byte letto dal testo) ed il contenuto della cella puntata da HL, che sappiamo contenere l'oramai famoso byte da «testare» (quello della DEFB).

Nel caso che non si abbia l'uguaglianza tra i due valori, allora si avrà il salto alla locazione **43D7H** dove troveremo un ulteriore salto alla routine di

gestione degli errori (posta all'indirizzo ØCC9H), in quanto in mancanza di «match» tra i due byte è richiesta l'interruzione del programma con la visualizzazione di un messaggio di errore, in questo caso un generico «Syntax error».

Nel caso invece che i due byte coincidano allora viene incrementato il contenuto di HL, che ora punterà (attenzione!) al byte e perciò all'istruzione successiva all'oramai «odiato» DEFB. La successiva istruzione **EX (SP), HL** ripristina il tutto, nel senso che ora nello stack abbiamo il corretto indirizzo di ritorno della subroutine 43C7H, mentre ora HL contiene ancora l'indirizzo dell'ultimo byte del testo MBASIC or ora analizzato: a questo punto è più che lecito aspettarsi l'incremento di tale coppia per poter analizzare il byte successivo di testo, come infatti accade...

Il successivo confronto con il valore 3AH, con conseguente «ritorno» se si è in condizione di «NC» (Not Carry) e viceversa con salto all'indirizzo **13ØAH** seguono la logica delle prime istruzioni della subroutine «13Ø5H», già analizzata nel n. 48 di MCmicrocomputer, in quanto sono praticamente le stesse istruzioni poste tra gli indirizzi 13Ø5H e 13ØAH: per tale ragione rimandiamo a quella puntata per i dettagli.

La subroutine 1D55H

Come si può vedere dal listato 2, tale routinetta è una subroutine vera e propria in quanto dotata non di una, ma di ben due istruzioni «RET»: si tratta di una routine che in base al valore contenuto nella cella di memoria ØA67H (il tipo di una variabile), setta secondo certe configurazioni i 4 flag principali e cioè il flag di zero (Z), il carry (C), il flag di segno (S) e quello di parità (P).

```
1D55H   LD A,(0A67H)
        CP 8
        JP NC,1D62H
        SUB 3
        OR A
        SCF
        RET
1D62H   SUB 3
        OR A
        RET
```

Listato 2 - Funzione che predispone i 4 flag principali dello Z80, a seconda del valore che rappresenta il «tipo» di una variabile.

Dal momento che i tipi di variabili sono quattro e cioè intero, stringa, reale o doppia precisione, sappiamo bene che basterebbero appena due flag, ma con quattro flag possiamo trattare bene anche altri casi che incontreremo fra breve.

Senza analizzare in dettaglio la subroutine, istruzione per istruzione, dal momento che consta di semplicissime operazioni logico-algebriche, andiamo invece a vedere la tabellina in cui riportiamo lo stato dei 4 flag citati in precedenza a seconda del tipo della variabile in esame, tipo codificato appunto con un valore posto nella cella di indirizzo ØA67H.

tipo	(0A67H)	Z	C	S	P
INTERO	2	0	1	1	1
STRINGA	3	1	1	0	1
REALE	4	0	1	0	0
D.PREC.	8	0	0	0	1

Analizziamo dunque quanto riportato dalla tabella, sapendo che a seguito della chiamata **CALL 1D55H** rimangono settati alcuni flag e resettati altri in base al contenuto della cella ØA67H.

In particolare il flag di Zero settato (condizione «Z») indicherà che la variabile in questione è di tipo stringa, la condizione «NC» indicherà la presenza di una variabile in doppia precisione, il flag di Segno settato individuerà una variabile intera, mentre infine la condizione «PO» (Parity Odd) segnalerà la presenza di una variabile in singola precisione.

A parte queste condizioni che consentono di individuare un singolo tipo tra i quattro, si possono come detto sfruttare combinazioni di tali condizioni per individuare ad esempio se una variabile è intera o reale, ma non in doppia precisione e tantomeno stringa: ciò si ottiene evidentemente con due test successivi.

Supponiamo ad esempio di voler discriminare il caso già citato di variabili reali o intere da quello di variabili di tipo stringa o in doppia precisione: ciò si ottiene testando dapprima la condizione di «Z», facendo saltare ad un'opportuna etichetta, e successivamente testando la condizione di «C» (carry settato), per l'appunto verificantesi quando le variabili non sono in doppia precisione. Il fatto di aver testato «prima» la condizione di «zero» elimina dalla seconda condizione il tipo stringa.

Invece se si desidera effettuare una certa routine se la variabile in esame è di tipo reale o in doppia precisione (genericamente in «floating point»), contrapponendo due routine nel caso di variabili intere o di tipo stringa, allora si potrà ad esempio testare subito lo stato del flag di segno, che esclude in caso di flag settato (condizione «M», «minus») la variabile intera e poi si effettuerà il test sul flag di zero, il quale, se settato, indicherà la presenza di una stringa e viceversa accomunerà variabili di tipo floating point, in quanto già dal primo test era stato escluso il tipo intero.

Al lettore attento non sarà sfuggito il fatto che le coppie di test indicate nei due esempi possono anche essere effettuate in ordine inverso: a dimostrazione di ciò vediamo cosa succede nel primo esempio se effettuiamo i test in ordine inverso.

Volendo discriminare in base allo stato del carry, ora si salterà alla routine di gestione delle variabili in doppia precisione nel caso in cui la condizione sia «NC».

Successivamente il test sul flag Z eliminerà il caso di variabile di tipo stringa.

Comunque non è un caso che abbiamo citato queste coppie di test: già dalla prima funzione che analizzeremo, troveremo applicazione di questi concetti.

Analisi della funzione SGN (x)

Si tratta di una funzione molto semplice, come ben sanno i lettori: essa fornisce un valore pari ad 1, Ø o −1 a seconda se l'argomento «x» risulta rispettivamente positivo, nullo o negativo.

Ovviamente è la funzione «segno», che va applicata a variabili di qualsiasi tipo, ad eccezione ovviamente delle stringhe.

Dal punto di vista dell'implementazione in Assembler, come si può vedere dalla parte iniziale del listato 3, la funzione in esame consta di due parti ben distinte: la prima nella quale viene chiamata la subroutine **2877H** e la seconda in cui in base al valore contenuto nell'accumulatore viene riempita la coppia HL da passare come parametro alla routine **29CDH.**

In particolare, tralasciando per un istante l'analisi della subroutine 2877H, diciamo che la coppia HL è posta rispettivamente ai valori ØØØØH, ØØØ1H e ØFFFFH (il quale ultimo vale −1 in logica complementata), rispettivamente se nell'accumulatore è posto un valore pari a ØØH, Ø1H e ØFFH, forniti appunto dalla 2877H.

L'analisi della routine **29CDH** ci dice che il valore posto in HL viene depositato nelle locazioni di memoria ØC04H e ØC05H (all'interno del Floating ACcumulator, FAC), mentre nella

locazione ØA67H (tipo della variabile) viene posto un valore pari a 2 e perciò indicante una variabile intera.

Prima di andare avanti facciamo due considerazioni: i valori citati di HL corrispondono appunto a quanto deve fornire la funzione SGN (x) ed in particolare il valore −1, espresso correttamente con ØFFFFH in quanto nell'interprete MBASIC si usa sempre per gli interi la logica complementata: infatti sappiamo che il massimo valore positivo rappresentabile in intero è 32767 mentre il minimo valore negativo è pari a −32768.

La seconda considerazione riguarda la routine chiamata al termine della routine principale: la **29CDH.** Tale routine assume per l'MBASIC un significato ben preciso in quanto è proprio la funzione MAKINT il cui entry point si trova nella ben nota «jump table» delle funzioni dell'MBASIC, riportata nel n. 38 di MC: ne abbiamo già parlato e perciò non insistiamo oltre.

Invece ora ci occuperemo della subroutine **2877H,** che ci servirà nel seguito nell'analisi di altre funzioni.

La subroutine 2877H

Sempre del listato 3, vediamo subito la chiamata alla subroutine **1D55H,** della quale abbiamo parlato poc'anzi.

Analizziamo dunque il comportamento della «2877H» in base allo stato dei 4 flag: se è settato il flag Z allora la variabile è di tipo stringa, nel qual caso si salterà all'indirizzo ØCE1H, all'interno della routine di gestione degli errori, per provocare la stampa del messaggio «Type mismatch error».

Successivamente, viene testata la condizione di «P» (Plus, flag di Segno a 0), che serve ad accomunare in un unico salto all'indirizzo 2836H i casi di variabili floating-point e cioè reali e in doppia precisione: analizzeremo nel seguito questo caso. Perciò, se i due test consecutivi non sono verificati, allora si tratta di una variabile intera e perciò si porrà in HL il contenuto intero del FAC (posto, come abbiamo appena visto, agli indirizzi ØC04H e ØC05H): se tale valore è nullo, allora in accumulatore avremo proprio Ø, con il quale usciremo dalla routine.

In caso contrario si pone in accumulatore il contenuto di H e si salta all'indirizzo **2840H**: qui troviamo alcune semplici istruzioni che ci consentono di ottenere in accumulatore un valore pari ad 1 se in H c'era un valore compreso tra ØØH e 7FH, mentre un valore pari ad FFH se in H c'era un valore compreso tra 80H e ØFFH. Pensandoci un attimo, troviamo che il valore di H è proprio la parte alta del valore intero che dovevamo analizzare: ecco che perciò un valore positivo (tra ØØØØH e 7FFFH) comporterà in uscita l'accumulatore posto ad 1, mentre un valore negativo (compreso tra 8ØØØH e ØFFFFH) comporterà un valore pari a ØFFH in accumulatore.

Tornando invece al caso delle variabili in floating-point, ci troviamo all'indirizzo **2836H**, dove viene caricato in accumulatore il contenuto della cella di memoria posta nel FAC e di indirizzo ØC07H, che nel caso di variabili reali ed in doppia precisione rappresenta l'esponente del numero in esame, secondo quanto abbiamo detto nel n. 47 di MC.

Se tale valore è nullo allora la variabile floating-point è considerata nulla ed ancora una volta si esce dalla subroutine con ancora una volta l'accumulatore correttamente azzerato.

Se questo non è il caso allora si carica in accumulatore la locazione ØC06H, rappresentante il byte più significativo della mantissa. Una successiva (e non molto chiara, lo confessiamo...) comparazione con il valore 2FH riporta poi ad eseguire la routine posta all'indirizzo 284ØH, già analizzata precedentemente.

Possiamo in definitiva riassumere le funzioni della subroutine 2877H nella seguente tabellina, nella quale, in funzione del valore iniziale del FAC, viene posto un opportuno valore in accumulatore e vengono settati i flag Z ed S:

val. FAC	Z	S	Accumul.	Note
nullo	1	Ø	Ø	Zero e Plus
positivo	Ø	Ø	1	Not zero e Plus
negativo	Ø	1	ØFFH	Not zero e Minus

In particolare, e ciò ci tornerà utile nella prossima puntata, se si esce dalla subroutine 2877H con la condizione di «Plus» allora il valore posto nel FAC era maggiore o uguale a Ø.

Con questo terminiamo la puntata e diamo l'appuntamento alla prossima in cui analizzeremo altre funzioni.

MC

```
286DH   CALL 2877H    ;funzione SGN(x)
        LD L,A
        RLA
        SBC A,A
        LD H,A
        JP 29CDH
2877H   CALL 1D55H
        JP Z,0CE1H
        JP P,2836H
        LD HL,(0C04H)
        LD A,H
        OR L
        RET Z
        LD A,H
        JP 2840H

2836H   LD A,(0C07H)
        OR A
        RET Z
        LD A,(0C06H)
        CP 2FH
2840H   RLA
        SBC A,A
        RET NZ
        INC A
        RET

29CDH   LD (0C04H),HL    ;MAKINT
        LD A,2
29D2H   LD (0A67H),A
        RET
```

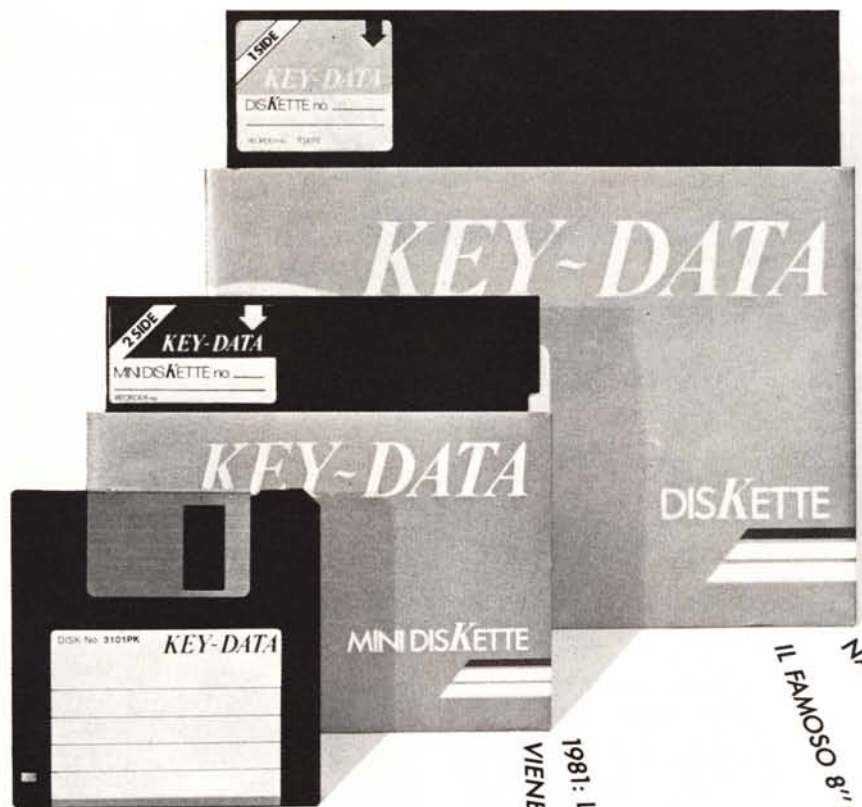
Listato 3 - Disassemblato della routine che implementa la funzione SGN (x).

gierre informatica

presenta

K KEY-DATA

L'EVOLUZIONE CHE GIRA NEL TEMPO



NEL LONTANO 1980 NASCEVA IL PRIMO FLOPPY-DISC

IL FAMOSO 8'' KEY-DATA

1981: LA RIVOLUZIONE DEI PERSONAL COMPUTERS

VIENE INTRODOTTO SUL MERCATO IL 5''¼ KEY-DATA

1984: CONTINUA L'EVOLUZIONE

NASCE IL PICCOLO MICROFLOPPY

IL 3,5'' KEY-DATA

I MINIDISKETTE 5''¼ KEY-DATA SONO DISPONIBILI IN VARIE COLORAZIONI

GI-ERRE INFORMATICA s.r.l.
42100 REGGIO EMILIA VIA UMBRIA 36/A TEL. 0522 38655 • 512345
70125 BARI VIA MONTE S. MICHELE 2/B TEL. 080 415975
95100 CATANIA P.ZZA GALATEA 2 TEL. 095 375222

INTERFACCE PER APPLE

Controller Doppio Drive	65.000
16K RAM	83.000
Language Card	101.000
80 Colonne Soft/Switch	118.000
8088 Card	690.000
Eprom Writer (16-64)	110.000
Prom Writer	493.000
Z/80 Card	61.000
RS-232 con cavo	100.000
Epson Printer e cavo	88.000
Grappler e cavo	98.000
Buffer 16K e cavo	265.000
Grappler + Buffer 16K	397.000
128K RAM	364.000
AD-DA 12 Bit/16 canali	504.000
AD Card	177.000
DA Card	298.000
IEEE-488	264.000
6809 Card	356.000
Communication Card	110.000
Super Serial Card	136.000
Pal Color Card	83.000
RGB (8 colori)	124.000
RGB II (16 colori)	194.000
Stereo Music Card	138.000
Scheda Parlante	87.000
Wild Card	87.000
Scheda orologio	124.000
6522 Card	155.000
Forth Card	131.000
I.C. Test Card	307.000
80 Colonne + 64K IIE	55.000
80 Colonne IIE	26.000
Adattatore x Drive IIC	20.000
Adattatore x Joystik IIC	14.000

DRIVE PER APPLE

Lisonic LS-39A	168.000
Chinon 051-AII	288.000
Mitac Ad-8	268.000
Chinon 360K + Contr.	470.000

STAMPANTI APPLE - IBM

— Star Gemini 10X (120 CPS./Dow Load)	535.000
— C.T.I. CPB-80 (130 CPS./2K Buffer/D. Load/Set IBM)	550.000
— Copal Sc-1200 L (120 CPS./N.L.Q./Set IBM)	650.000
— KDC-FT 5002 NEW (120 CPS./1K Buffer/Down Load/NQL/Set IBM)	690.000
— C.I.T. CPB-136 (130 CPS/2K Buffer/D. Load/Set IBM)	860.000
— Copal SC-5500I (180 CPS/132 Col./3K Buffer/D. Load/N.L.Q./Set IBM)	1.150.000

INTERFACCE PER PC/XT IBM

Controller 4D. + Cavo	196.000
Printer Card	112.000
Color Graphic	280.000
Color Graphic + Printer	370.000
Monoc. Gaph. (Hercules II)	320.000
Multifunction 256K	274.000
Multifunction 384K	364.000
AD-DA (12 Bit - 16 Can.)	499.000
512 RAM (Ø RAM)	166.000
RS-232	160.000
Game I/o	88.000
Eprom Writer (16-256)	430.000
8255	290.000
Rete Locale I-NET	980.000

DRIVE PER PC/XT IBM

Chinon 502	299.000
Chinon 502L	347.000
Matsushita 561 1 MB	480.000

INTERFACCE E DRIVE X AT

AT Controller	460.000
AT Parall/Serial Card	280.000
AT Multifunction 2,5 MB	680.000
AT H. Disk Controller + 2FDD	1.380.000
Hard Disk Teac 10 MB (senza contr.)	1.320.000
Hard Disk Seagate 20 MB (senza con.)	1.690.000

AT COMPATIBILE

Versione Base: Main Board ØK espandibile ad 1 M.B., alimentatore 200 W, Cabinet in metallo, tastiera **L. 3.950.000**

II E COMPATIBILE + PAD NUMERICO

128K RAM
+ 80 Colonne

II E compatibile al 100% dotato di comodo Pad numerico. Viene fornito completo di scheda 80 colonne + 64K **L. 760.000**

II E COMPAT. CON TASTIERA SEPARATA

128K RAM
+ 80 Colonne

II E compatibile al 100%. Versatile realizzazione con tastiera separata, intelligente, Pad numerico, Tasti funzione definibili, Autorepeat e «Bip» disinseribile. Possibilità di inserimento di N. Due Drive Slim all'interno del Cabinet. **L. 920.000**

PREZZI IVA ESCLUSA

LA CASA DEL COMPUTER
DISK-DRIVE · STAMPANTI · COMPUTER · MONITOR
IMPORTAZIONE DIRETTA

0587 212.312

VIA MISERICORDIA, 84 - 56025 PONTEDERA (PI)

FRA TUTTI I COMPATIBILI IL NOSTRO GIRA PIÙ VELOCE

PC/XT TURBO

L. 1.680.000

Clock 6,67-4,77 MHz
Main Board Esp. 640K

NOVITÀ

N. 1 Drive DS/DD 360K controller,
Main Board ØK espandibile A 640K,
Alimentatore 130 W, Tastiera K5 S

PC/XT STANDARD (4,77 MHz)

L. 1.420.000

Configurazione come sopra ma con Main Board 128K espandibile a 256K

* * * Per le interfacce video vedere listino* * *

Monitor Philips Monocr. x IBM	L. 227.000
Monitor Cabel MR Colori x IBM	L. 439.000
Monitor Philips HR Colori x IBM	L. 690.000

ABBIAMO PRONTA CONSEGNA:
— Stampanti 120-130-180 CPS — Modem
— Accoppiatori acustici — Monitor
— Hard Disk 10-20 MB — STREAMER 20 MB
— Plotter — Digitizer
RICHIEDETECI LO «SPECIALE PROMOZIONE»

DATAFLEX

PROFESSIONAL 5¼

CON BOX IN PLASTICA OMAGGIO!!!
SCONTI PER QUANTITÀ

SINGOLA F. - DOPPIA D.		DOPPIA F. DOPPIA D.	
200 Pezzi	2.230	200 Pezzi	2.870
100 Pezzi	2.350	100 Pezzi	3.150
30 Pezzi	2.550	30 Pezzi	3.400

- I dischetti dataflex sono prodotti da uno dei più grossi fabbricanti americani che garantisce l'altissima qualità ed affidabilità.
- Uno speciale ed esclusivo strato «MUlticot» protegge la superficie dall'usura del contatto con le testine garantendo minimo ben 10.000.000 di passaggi!!!
- La sicurezza dei Vs. dati è assicurata dall'ineccepibile supporto magnetico di primissima qualità.

DATO L'INSTABILE MERCATO DEI CAMBI PREGASI TELEFONARE PER CONFERMA PREZZI E DISPONIBILITÀ
— RICHIEDETECI IL CATALOGO — SCONTI AI SIG. RIVENDITORI

i trucchi del CP/M

a cura di Pierluigi Panunzi

Il CCP (Console Command Processor)

Dopo aver parlato per parecchie puntate del BDOS, ecco che ci occuperemo in questa di un'altra parte costituente il CP/M, il modulo detto CCP, dalle iniziali del nome Console Command Processor. Tale nome indica appunto qual è la sua funzione principale: quella di processare i comandi provenienti dalla console.

In termini tecnici è in un certo senso l'«interfaccia» tra l'operatore che digita un comando tramite la tastiera ed il nucleo principale del CP/M costituito dal ben noto BDOS: un comando digitato e mostrato lettera per lettera sullo schermo video verrà eseguito appunto dal CCP non appena avrà ricevuto, tra gli altri, il carattere CR (Carriage Return).

Innanzitutto andiamo a vedere in quale parte della memoria risiede il CCP rispetto al BDOS ed al BIOS: osservando la figura 1 (tratta dal «solito» «The CP/M Programmer Handbook» di A. Johnson/Laird) ritroviamo quanto già detto in scorse puntate della rubrica e cioè che, partendo dall'indirizzo più basso (0000H) si trovano 256 (0100H) byte riservati al CP/M, poi tutta la memoria libera da usarsi per i programmi applicativi, poi ancora c'è il nostro CCP (ampio 0800H e cioè 2 Kbyte), seguito dal BDOS (ampio 0E00H e cioè 3.5 Kbyte) ed infine dal BIOS, di ampiezza dipendente dal sistema in esame: in particolare gli indirizzi riportati in esadecimale ed in decimale sono solamente indicativi e servono principalmente a dare un'idea dell'ordine di grandezza dei moduli in gioco.

Già sappiamo inoltre che all'atto del «Warm boot» il CCP viene ricaricato dal dischetto subito al di sotto del BDOS.

Questo perché parecchi programmi deliberatamente occupano la zona di memoria a lui riservata, per i propri scopi: sono 2 Kbyte che a volte possono essere molto utili.

Perciò quando uno di questi programmi (detti «transienti») termina l'elaborazione, subito passa il controllo al BIOS che a sua volta ricarica il CCP dal dischetto e a sua volta ancora passa il controllo proprio al CCP.

Come ben sanno gli utenti di sistemi dotati di CP/M, è proprio dal CCP che viene generato il prompt «A>» indicante che il CCP è in attesa di comandi, formati dal nome del programma da eseguire, opzionalmente seguito dalla cosiddetta «Command tail» e cioè dagli eventuali parametri da passare al programma.

In realtà quello che abbiamo indicato genericamente come «nome del programma da eseguire» può essere sia il nome di un file residente sul dischetto oppure uno dei comandi frequentemente usati e inglobati all'interno del CCP.

Digitando il nome di uno di questi ultimi il CCP non avrà bisogno perciò di caricare un file dal disco; in caso non si tratti di tali comandi interni allora si avrà la ricerca di un file avente tale nome, seguito dall'estensione «.COM».

Nel caso in cui il CCP non trovasse il file nel dischetto allora segnalerà il fatto sul video replicando il nome del comando, seguito da «?».

I comandi interni

Dal momento che si tratta di argomenti ormai ben noti ai lettori, in quanto il CP/M stesso è in auge da parecchi anni, di certo non pretendiamo di scoprire cose nuove: ad esempio chi non sa che con il comando DIR si ottiene la directory del disco?!

Invece intendiamo segnalare caratteristiche meno note o appariscenti (è lo spirito della rubrica...), che magari possono trarre in inganno l'utente inesperto o disattento.

Cominciamo dunque dal primo comando, che possiamo genericamente indicare come **X:**, dove per «X» si intende la lettera identificatrice di un'unità logica a dischi.

Sappiamo che all'accensione della macchina, l'unità logica di default è la «A:»: per passare da tale unità all'unità «B:», bisogna appunto digitare **B:** e RETURN.

Mentre nel caso di un dischetto il passaggio dall'unità «A:» alla «B:» è praticamente istantaneo, ciò può non accadere nel caso si lavori con un sistema dotato di hard-disk, nel senso che in alcuni casi ci vuole un paio di secondi per riottenere il prompt e perciò la possibilità di eseguire altri comandi.

La spiegazione di tale differenza deriva dal fatto che, quando decidiamo di cambiare unità logica, il BDOS deve andare a leggersi l'intera directory del nuovo disco per costruirsi il ben noto «allocation vector» relativo a quel disco: perciò mentre in un dischetto i file sono in genere pochi, in un hard-disk possono risiedere anche parecchie centinaia di programmi con una directory molto estesa.

Il secondo comando, arcinoto, è **DIR**, che, senza necessità di essere eseguito da command tail del tipo «*.*», consente di vedere quali file sono contenuti nel disco, eccettuati quelli aventi l'attributo di «system», che vengono completamente ignorati: a tale scopo è molto più utile il programma STAT che consente su richiesta di conoscere le caratteristiche logiche di ogni file (quali ad esempio l'estensione), come pure di cambiare lo stato di un file da «system» a normale.

Altro comando è il famoso e terribile **ERA**, terribile in quanto non ammette ripensamenti immediati: infatti cancella il file o i file indicati subito dopo, senza però chiederne una conferma (cosa che accade invece nei sistemi operativi più recenti).

Unico caso in cui si mostra clemente è quello in cui l'utente decidesse di cancellare tutti i file contenuti sul dischetto con il comando «**ERA *.***»: in tal caso è il CP/M stesso che tenta di fermare l'insano gesto dell'utente, chiedendo «ALL (Y/N)?» quasi a supplicare il programmatore di evitare l'ecatombe. Il fatto è che se ad esempio sul disco esiste un solo file e noi vogliamo cancellarlo, svuotando così il disco, il nostro sistema operativo non si scompone minimamente, senza nemmeno preoccuparsi che per effetto della cancellazione non esistano più file sul disco: qual era dunque la necessità di fermare la mano assassina che si abbatteva su tutti i file del disco, quando l'utente esperto può abilmente cancellare i file senza far scomporre il CP/M?

Per quanto riguarda il comando **REN** ricordiamo con una certa dose di disappunto che non è possibile usare nomi ambigui tanto per il «nome nuovo» quanto per il «nome vecchio», tramite l'uso di «*» e «?» all'interno dei nomi stessi: per ottenere ciò si deve per forza usare un programma ad

hoc, «transiente», ed un esempio si può trovare nel numero 33 di MCmicrocomputer, sempre nell'ambito di questa rubrica. Per quanto riguarda il comando **TYPE** ricordiamo che serve a mostrare sullo schermo video il contenuto di un file di testo e cioè contenente caratteri ASCII: questo perché nel caso si esegua il TYPE di un file di comando (e perciò contenente un programma) allora tutti i caratteri non ASCII verrebbero interpretati come caratteri di controllo, con conseguenze a volte catastrofiche nel senso che si è in genere costretti a spegnere completamente il computer.

Per fermare l'output su video si utilizza il classico CONTROL-S, mentre per riprendere l'output basta un carattere qualunque anche se quello «ufficiale» è il CONTROL-Q, che assieme al precedente, costituiscono i ben noti «X-OFF» e «X-ON» dell'omonimo protocollo di rice-trasmissione.

Il comando **SAVE** invece è allo stesso tempo il più atipico e, se vogliamo, strano nel senso che, come è noto, serve a salvare su disco il contenuto di «n» pagine di memoria (256 byte) a partire dall'indirizzo Ø1ØØH e cioè un programma «transiente».

È molto utile ad esempio quando si vogliano «customizzare» i programmi, dopo aver ad esempio modificato alcune locazioni di esso tramite il programma DDT o ZSID, rispettivamente «della casa» (orientato perciò all'8080) e della Zilog (orientato ovviamente allo Z80): come il lettore attento si ricorderà, tale prassi è usata parecchie volte nel corso delle varie puntate della rubrica «I trucchi del CP/M». Ricordate le modifiche al WordStar?

Per quanto riguarda l'ultimo comando, **USER**, i lettori concorderanno nell'assegnargli il premio per il «comando meno utilizzato» e dunque «più inutile»... Ci domandiamo infatti quale utente si sia mai dato la classica zappa sui piedi impelagandosi in lunghe ricerche tra i vari «user» (vedasi a tale proposito il programma SEARCH pubblicato sul numero 48 di MC), a meno che non abbia a disposizione un hard disk ed abbia potuto così creare delle sotto-directory, per non naufragare nelle centinaia di programmi dei quali abbiamo parlato a proposito del comando DIR: comunque rimane il fatto che programmi appartenenti ad un certo «user» non possono essere eseguiti nell'ambito di un altro user, a meno di non ricopiarli anche nel nuovo user, con conseguente spreco di spazio.

Terminata dunque questa carrellata sui comandi interni del CP/M, andiamo ad analizzare le possibilità di editing di una linea di comando, delle quali forse la più nota ed usata è il CONTROL-H o Back-space.

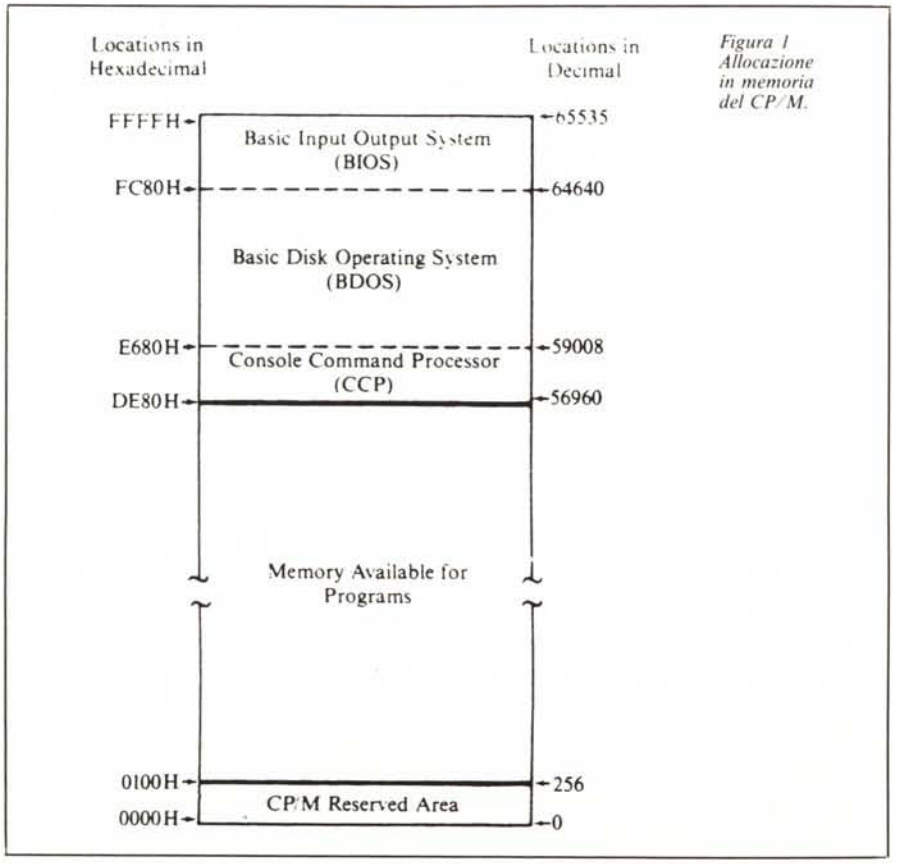


Figura 1 Allocazione in memoria del CP/M.

L'Editing di una linea di comando

Quando noi impostiamo da tastiera un certo comando, contemporaneamente il CCP memorizza ogni carattere digitato in un apposito buffer (posto proprio nelle primissime locazioni della zona di memoria riservata al Console Command Processor), in attesa che l'operatore dia il consenso a quanto impostato, tramite il classico RETURN.

L'editing di linea consiste nella possibilità da parte dell'operatore di inviare, invece di semplici caratteri alfanumerici, dei ben specifici «caratteri di controllo», ottenuti premendo contemporaneamente il tasto «Control» (di solito indicato con CTRL) ed una lettera: sullo schermo, a parte i casi che citeremo tra breve, e cioè nel caso in cui il carattere di controllo non abbia un particolare significato, si otterrà la stampa di un «accento circonflesso» («^»), che gli anglofoni chiamano «caret» o «cusp», seguito dalla lettera digitata. Tra l'altro il carattere «^» è quello usato ad esempio nel Basic per effettuare l'elevamento a potenza e può sorgere la tentazione di digitare un CONTROL-^, al che il CP/M non si scompone visualizzando correttamente i caratteri «^^».

Iniziamo dunque ad analizzare i vari caratteri di controllo che hanno un certo significato per il sistema operativo. Il primo e più noto è **CONTROL-C** (denominato anche ETX): inserito come primo carattere di una linea di comando forza un «warm boot», con conseguente caricamento del CCP: come gli utenti CP/M ben sanno, questa è un'operazione obbligatoria ogni volta che si cambia il dischetto, se non si vogliono ottenere i famosi «BDOS error on A: R/O».

Il secondo carattere di controllo, in ordine alfabetico, è il **CONTROL-E**, raramente usato nei moderni terminali dotati di «wrap-around» automatico a fine linea, mentre è indispensabile con alcuni tipi di teletype che viceversa ricomincerebbero a scrivere nella stessa linea.

Del carattere **CONTROL-H** praticamente non c'è nulla da dire tranne che «intelligentemente» riconosce se deve cancellare un singolo carattere, nel caso di una lettera ad esempio, oppure due caratteri nel caso si tratti di un carattere di controllo: in caso contrario si otterrebbero linee contenenti strani «^».

Per il CCP abbiamo detto che è significativo il RETURN indicante il consenso ad un certo comando impostato: tale carattere si ottiene automaticamente premendo il tasto «RETURN» che invia di solito un **CONTROL-M** seguito da un **CONTROL-J**: inviati separatamente e cioè pre-

mendo CTRL e la lettera M o J, agiscono come se si fosse premuto il RETURN. Quello che forse è poco noto è il vero significato dei due caratteri di controllo: dal momento che il RETURN li invia accoppiati ha comportato il fatto che non se ne ricorda più la singola funzione.

Il carattere **CONTROL-J**, detto comunemente «**LINE FEED**», ha il compito di far avanzare lo schermo o la stampa di una linea, mantenendo inalterata la posizione del cursore nell'ambito della linea stessa: tale funzione si ritrova in parecchi computer associata alla «freccia in giù», che di certo non deve riportare a capo il cursore!

Per quanto riguarda il carattere **CONTROL-M**, detto anche «CR» o «Carriage Return», invece dobbiamo dire che è lui che fa ritornare il carrello («carriage») o il cursore all'inizio della linea, senza però abbandonare la linea su cui si stava scrivendo per passare alla successiva.

Tutto questo l'abbiamo verificato personalmente nelle teletype: se l'operatore digita qualcosa seguito dal «RETURN», non otterrà altro che il ritorno del carrello di stampa all'inizio della linea con il risultato di sovrascrivere i caratteri appena scritti.

Viceversa premendo il tasto «LINE FEED» (in alcuni casi «NEW LINE») il carrello non ritorna certo all'inizio della riga successiva, ma proprio sotto all'ultima posizione raggiunta.

Del carattere **CONTROL-P** ricordiamo (a parte la sua denominazione ASCII di «DLE» e cioè «Data Link Escape») che serve come «interruttore software» per l'attivazione e la disattivazione della stampante, che deve essere presente ed in stato di «ON LINE», altrimenti il computer rimane inchiodato, sensibile solo al reset.

Del carattere **CONTROL-R** parleremo fra breve, mentre ora ricordiamo che **CONTROL-Q** (X-ON) e **CONTROL-S** (X-OFF) servono come già detto a far partire ed interrompere una generica trasmissione.

Per quanto riguarda il **CONTROL-U** (NAK, Negative AcKnowledge) ed il **CONTROL-X** (CAN, CANcel line) comunicano al CCP il ripensamento dell'operatore riguardo a quanto stava digitando: con il primo il CCP visualizzerà un carattere «#», imposterà automaticamente la coppia «RETURN-LINE FEED» ed alcuni spazi bianchi, dopodiché attenderà una nuova linea di comando. È importante notare che la vecchia linea non viene cancellata. Invece con il **CONTROL-X** si ha una cancellazione fisica della linea finora digitata ed è più utile nei terminali video.

Un altro carattere di controllo, non molto utile dato che serve praticamente solo a confondere le idee, è il cosiddetto «**RUB**», «**RUBOUT**», «**DEL**» o «**DELETE**» che dir si voglia (a seconda di quale è stata la scelta del costruttore del terminale): in particolare cancella l'ultimo carattere impostato «senza cancellarlo» dallo schermo, ma addirittura ripetendolo, permettendo così di ottenere abominevoli sgorbi.

Ad esempio basta digitare il classico «PIPPO» seguito da due «rubout» e magari da altre due lettere («AA») per ottenere un comando che si legge «PIPPOOPAA»: ecco che arrivati a questo punto arriva il salvataggio grazie all'ultimo carattere di controllo, il **CONTROL-R** di cui abbiamo solo accennato prima, che effettua un salutare «refresh» della linea digitata, dopo aver impostato un «#» sulla linea incriminata ed essere passato a nuova linea e posizionando correttamente il cursore al punto in cui possiamo continuare a digitare caratteri.

C'è da dire che in alcune tastiere il carattere «RUBOUT» o «DELETE» può anche mancare oppure, ma è un caso particolare, può essere associato alla funzione di cancellazione dello schermo video.