A cura di Pierluigi Panunzi


# L'istruzione di assegnazione «LET»

*Nelle ultime due puntate di questa rubrica abbiamo dato ai lettori «un assaggio» di quanto ci può fornire, sotto forma di spunti e di trucchi, l'interprete MBASIC, dal punto di vista della sua costituzione «fisica», «interna»: questa volta passeremo a qualcosa di più sostanzioso.*

*Come i lettori già sanno e come d'altro canto apparirà evidente da quanto diremo, non ci interessa assolutamente scendere nei minimi dettagli (... fino all'ultimo RET di subroutine chiamate da subroutine a sua volta chiamate da subroutine...), ma bensì ci interessa fondamentalmente seguire un ideale flowchart lungo il quale avere sempre presente (si spera...) ciò che il programma fa.*

Questa volta, dunque, analizzeremo quella che «sulla carta» è l'operazione più semplice e cioè l'«assegnazione» di un certo valore ad una variabile:

[LET] <var> = <espressione>

dove il comando LET, come sappiamo, è opzionale, ma in questo caso ci è utile in quanto ci permette di individuare l'entry point della routine assembler, entry point che troviamo (nella solita tabella pubblicata nel n° 38) essere pari a 1518H.

Prima di andare ad analizzare il «disassemblato» della routine in esame, facciamo una breve pausa di riflessione, per vedere in quale contesto ci troviamo e che cosa dobbiamo aspettarci.

Per ottenere ciò dobbiamo per un istante immedesimarci in quello che si chiama «analizzatore sintattico-lessicale» e perciò andando a leggere, passo dopo passo, la generica linea del nostro programma scritto in MBASIC.

Dapprima supponiamo di trovare la parola-chiave «LET», per cui sappiamo che si tratterà di un'istruzione di assegnazione: ora dovremo incontrare il nome di una variabile, nome che deve cominciare per lettera e terminare ad esempio con il carattere «%» se la variabile deve essere intera.

Subito dopo dobbiamo trovare il segno « = », altrimenti il resto della linea non avrebbe senso per noi, seguito dal quella che abbiamo precedentemente indicato come una generica <espressione>.

Su quest'ultima e sul modo di trattarla da parte dell'MBASIC si potrebbe scrivere un libro, tante sono le caratteristiche che la contraddistinguono. Ne citiamo un paio, tanto per fare un esempio: il numero ed il tipo dei vari operandi, con le eventuali conversioni del caso (con le ovvie segnalazioni del caso, se si somma un numero ad una stringa...), oppure la precedenza tra gli operatori (verrà eseguito prima un elevamento a potenza o un OR?!).

Una volta calcolata la nostra espressione, per lunga che sia, il programma in linguaggio macchina dovrà provvedere a memorizzarne il valore nelle celle di memoria precedentemente riservate alla variabile in questione.

In realtà la dizione «per lunga che sia» non è esatta, in quanto potrebbe capitare qualche caso in cui l'espressione è così complicata che neanche il computer riesce ad analizzarla tutta: siamo in un caso limite, ma forse non è noto a tutti che in tali improbabili casi l'MBASIC emette il messaggio «Out of memory».

Prima di passare definitivamente all'analisi, fissiamo l'attenzione sul fatto della memorizzazione del valore dell'espressione: nel numero 42 di MC abbiamo parlato in dettaglio delle modalità di codifica dei vari valori numerici (interi, reali, stringhe o in doppia precisione) da parte dell'MBASIC. Ora troveremo conferma di quanto detto.

In particolare il risultato di un'espressione viene posto nel cosiddetto FAC (Floating point Accumulator), del quale abbiamo già parlato nel numero 43 e che ora troveremo in memoria a partire dall'indirizzo 0C00H fino a 0C07H: a seconda del «tipo» della variabile sappiamo che il suo contenuto assumerà una differente dislocazione. Per comodità dei lettori riassumiamo in tabella 1 il contenuto del FAC a seconda del tipo della variabile interessata.

Andiamo dunque ad analizzare la routine «di assegnazione», a partire dall'indirizzo 1518H, seguendo il listato.

## L'istruzione LET

Dando un'occhiata rapida al listato, del quale abbiamo tralasciato una parte relativa alle stringhe e che richiede conoscenze più approfondite, troviamo in tutto 5 «CALL» ad altrettante subroutine di non difficile individuazione tanto è vero che basta analizzarle singolarmente per comprenderne il funzionamento. Nell'ordine troviamo subito una CALL 379CH che da sola si preoccupa di analizzare il testo per ottenerne il nome di una variabile (segnalando errori nel caso che il nome non cominci per lettera, ad esempio se scriviamo "&X = 5") e poi di controllarne l'eventuale esistenza, viceversa allocando un certo spazio in memoria dove poi andare a deporre il valore dell'espressione.

Tabella 1

| Indirizzo | Intero | Stringa | Reale | Doppia Prec. |
|---|---|---|---|---|
| 0C00H | — | — | — | LSBM |
| 0C01H | — | — | — | . . . |
| 0C02H | — | — | — | . . . |
| 0C03H | — | — | — | . . . |
| 0C04H | LSB | PUNT L | LSBM | . . . |
| 0C05H | MSB | PUNT H | . . . | . . . |
| 0C06H | — | — | MSBM | MSBM |
| 0C07H | — | — | ESP | ESP |

significato dei simboli:

| | |
|---|---|
| LSB,MSB | parte bassa e parte alta di numero a 16 bit |
| PUNT L | parte bassa di un puntatore |
| PUNT H | parte alta di un puntatore |
| LSBM | byte meno significativo della mantissa |
| . . . | byte successivi della mantissa |
| MSBM | byte più significativo della mantissa |
| ESPE | esponente del numero reale o in doppia precisione |
| — | byte non significativo, non usato |

*Utilizzazione del Floating Accumulator a seconda del tipo della variabile.*

Detto così la subroutine in esame sembra quasi banale, mentre viceversa risulta alquanto complessa: basti pensare che se la variabile non esisteva già precedentemente, allora l'interprete dovrà farle spazio in memoria (in una zona ben precisata) spostando altre variabili già esistenti, preoccupandosi di mantenere l'ordinamento alfabetico delle variabili stesse, nonché controllando volta per volta che la memoria rimanente lo consenta.

Ben sanno i lettori che programmi lunghi e con molte variabili procurano molto spesso problemi del genere, segnalati da un perentorio «Out of Memory Error».

Subito dopo troviamo l'oramai ben nota chiamata alla subroutine 43C7H, che si assicura che nel testo si trovi proprio quel carattere posto subito dopo la chiamata («=» nel nostro caso). Ecco che con tre istruzioni (!) abbiamo già analizzato la variabile e l'«=»: ora quel che manca è il calcolo dell'espressione, non prima di «averne creato le premesse», ed infine la memorizzazione finale del risultato.

Tra le varie locazioni di memoria settate durante lo svolgimento della subroutine 397CH («cerca e crea una variabile»), c'è la 0A67H, usata poi un'infinità di volte, dove si troverà il «tipo» della variabile considerata, nel nostro caso la variabile a sinistra dell'«=»: in particolare, ma per il lettore attento non è una novità, si ha la corrispondenza seguente:

Tabella 2

| valore | tipo della variabile |
|---|---|
| 2 | intero |
| 3 | stringa |
| 4 | singola precisione |
| 8 | doppia precisione |

che ricalca ovviamente l'occupazione in byte della singola variabile. Ecco dunque che troviamo, scorrendo il listato, e saltando talvolta alcune istruzioni,

— la lettura della cella 0A67H e successivo salvataggio nello stack;

— la chiamata alla subroutine 19F3H (eccola qui! questa subroutine effettua il calcolo di <espressione>, ponendo il risultato nel FAC ed il «tipo» del risultato ancora una volta in 0A67H);

— il ripristino dell'accumulatore dallo stack con conseguente salvataggio del valore del registro B, per effettuare la comparazione dei due «tipi», quello della variabile e quello dell'espressione;

— nel caso in cui il «tipo» della variabile ed il tipo dell'espressione non fossero uguali (ma sempre congruenti, ovviamente), allora si effettua la chiamata alla subroutine 1F9DH, preposta appunto alla conversione del risultato al «tipo» richiesto: facciamo subito un esempio per chi non avesse ben compreso di cosa stiamo trattando in questo punto.

Mentre sappiamo bene che il computer segnalerà errore se scriviamo un'assegnazione del tipo

A% = «PIPPO»

in quanto tentiamo di associare una stringa ad una variabile intera (ma analogamente accadeva se la variabile era reale o in doppia precisione), viceversa sappiamo che è perfettamente lecita un'assegnazione del tipo

A% = 49.287

oppure

B# = 0

oppure ancora

X# = Y + Z% – 4 / F#

in quanto in questi casi si ha una conversione tra quelle permesse e cioè

intero - reale
intero - doppia precisione
reale - intero
reale - doppia precisione
doppia precisione - intero
doppia precisione - reale

Effettuata o meno la conversione, a questo punto abbiamo il risultato nel FAC, a partire da un indirizzo differente a seconda che si tratti di variabile in doppia precisione o altro: infatti si può vedere dalla tabella 1 che solo nel caso di variabile in doppia precisione sono significativi i byte compresi

```
1518H   CALL 397CH        ; gestione della  variabile
        CALL 43C7H        ; scan del testo
        DEFB '='          ; alla ricerca dell'"="
        EX DE,HL
        LD (AA7H),HL
        EX DE,HL
        PUSH DE
        LD A,(A67H)       ; tipo della variabile
        PUSH AF
        CALL 19F3H        ; calcolo dell'  espressione
        POP AF
        EX (SP),HL
        LD B,A            ; tipo della variabile
        LD A,(A67H)       ; tipo dell'  espressione
        CP B              ; confronto tra i due tipi
        LD A,B
        JP Z,153DH        ; salta se uguali
        CALL 1F9DH        ; conversione di tipo
        LD   A,(A67H)

153DH   LD DE,ØC04H       ; punta a FAC-3
        CP 5              ; test se in doppia precisione
        JP C,1548H        ; se no, salta
        LD DE,ØC00H       ; punta a FAC-7
1548H   PUSH HL
        CP 3              ; test se è stringa
        JP NZ,157FH       ; se no,salta
        LD HL,(ØC04H)     ; stringa: puntatore allo
        PUSH HL           ; "string descriptor"
        INC HL            ; punta all'indirizzo della stringa
        LD E,(HL)
        INC HL
        LD D,(HL)         ; prosegue l'elaborazione nel
                          ; caso della stringa
         ...
157FH   CALL 28BAH        ; ritorno comune: memorizzazione
        POP DE
        POP HL
        RET               ; ritorno all'analizzatore
```

*Listato 1 - La routine relativa all'istruzione LET.*

tra 0C00H e 0C03H, mentre negli altri casi (intero, stringa o reale) il primo byte utile è posto in 0C04H. Ecco che perciò, volendo porre nella coppia DE l'indirizzo iniziale «utile» di FAC, allora basterà inizializzare DE al valore 0C04H eventualmente da cambiare in 0C00H nel caso in cui la cella 0A67H (ancora lei!) contenga un valore maggiore di 5...

Stupisce a questo punto il fatto che si effettui per questo scopo la comparazione con il valore 5, per poi saltare nel caso in cui il Carry sia settato, invece di testare semplicemente se l'accumulatore vale 8 (doppia precisione), nel qual caso si salta la variazione del valore di DE: diciamo questo in quanto sembrerebbe possibile l'esistenza di variabili di tipo 5, 6 e 7, apparentemente in contrasto con quelle che sono le specifiche dell'MBASIC... Ne riparleremo forse in seguito.

Si tratta della quinta subroutine usata da questa parte di programma e precisamente quella posta all'indirizzo 28BAH, che riportiamo nel listato 2.

## La routine 28BA di trasferimento

Ecco finalmente una routinetta semplice semplice che sposta un certo numero di byte da una zona all'altra della memoria, rispettivamente puntate da DE e da HL: analizzandola si «scopre» ancora una volta che l'MBASIC è scritto «in 8080» e non certo «in Z80».

Con quest'ultimo si potrebbe usare proficuamente l'istruzione LDIR al posto dei vari caricamenti di A e relativi scaricamenti, incrementi dei puntatori, decrementi in B e salto (orrore!) se B è ancora diverso da 0 (quant'è comoda invece la DJNZ...). Comunque l'importante è che una routine funzioni bene: solo se poi ci fossero problemi di spazio, allora si potrebbe tentare un certo raffinamento.

```
28BAH   LD A,(0A67H)    ; tipo della variabile
        LD B,A          ; è anche il numero di byte
28BBH   LD A,(DE)       ; prelievo dal FAC
        LD (HL),A       ; memorizzazione nella zona apposita
        INC DE          ; incrementopuntatore
        INC HL          ; incremento puntatore
        DEC B           ; decremento contatore
        JP NZ,28BBH     ; è finito? salta, se no
        RET             ; ritorno al programma chiamante
```

*Listato 2 - La routine di spostamento di byte dal FAC (Floating ACcumulator) alla zona riservata alle variabili.*

A questo punto abbiamo la coppia DE che punta all'inizio utile del Floating Accumulator, mentre viceversa HL punterà (si vede ciò scorrendo a ritroso il listato e soffermandosi sulle istruzioni non analizzate), dicevamo HL invece punta alla zona di memoria predisposta dall'MBASIC per le variabili e gli array, subito a ridosso dell'ultima istruzione del nostro programma in Basic: forse non tutti sanno che tale zona viene automaticamente spostata ogniqualvolta venga inserita una nuova linea di programma. Anche se ciò potrebbe non sembrare ovvio, in caso di aggiunta di linee, tale zona viene completamente distrutta e non semplicemente traslata, dal momento che ciò che varia è solo l'indirizzo iniziale di tale zona: evidentemente il fatto di inserire una nuova linea significa sconvolgere il più delle volte il nostro programma, per cui non ha certo senso mantenere i valori vecchi delle variabili. Dunque ora, se il tipo della variabile è 3 (stringa) allora verrà eseguita una sequenza di istruzioni che abbiamo volutamente trascurato ed indicate da «...», in quanto ci avrebbero portato troppo lontano; negli altri casi (intero, reale o doppia precisione) invece si salta tale zona e si va a chiamare una certa subroutine.

## Il caso delle stringhe

Abbiamo detto già che il caso delle stringhe l'abbiamo tralasciato in quanto richiede ulteriori approfondimenti. In questa sede diciamo che nel FAC e precisamente agli indirizzi 0C04H e 0C05H troviamo un puntatore (cioè una word, formata da 2 byte) al cosiddetto «string descriptor» e cioè una terna di byte così configurata:

1° byte — lunghezza della stringa
2° e 3° byte — puntatore alla stringa

Come si vede, perciò, la stringa generica non può essere più lunga di 255 caratteri, ma viceversa può anche avere «lunghezza nulla», come è ben noto ai programmatori. Inoltre si vede che il secondo ed il terzo byte contengono l'indirizzo fisico a cui è allocata la stringa, in una zona di memoria compresa tra la zona riservata alle variabili ed il cosiddetto «Top of Memory», cioè l'inizio del BDOS.

Senza addentrarsi in discorsi che ci porterebbero molto lontano, ricordiamo che questa zona riservata alle stringhe richiede di tanto in tanto la cosiddetta «garbage collection» e cioè la pulizia da tutti i residui di vecchie stringhe in formazione, oramai inutilizzate, ma che occupano spazio.

## Alcune notizie sui nomi delle variabili

Concludiamo questa puntata con alcune notiziole riguardanti le variabili dell'MBASIC, forse già note ai lettori, ma che è senza dubbio utile ricordare, visto il proliferare di «dialetti» che non sempre mantengono le convenzioni dell'originale.

Innanzitutto le variabili devono iniziare per lettera (e fin qui siamo d'accordo) e possono avere un nome lungo al massimo 40 caratteri: tali caratteri possono essere le lettere, i numeri, nonché (e chi lo sapeva?! anche noi siamo rimasti sorpresi) il «punto decimale» ('.').

Ecco dunque che una variabile si può senz'altro chiamare

PIPPO.PIPPO

e per nessun motivo potrà mai essere confusa con un'altra variabile PIPPO: se vogliamo, possiamo anche «abundare» e creare variabili che si chiamano

A.A.A.A.A.A.X.X.X.X.X.

lunghe da digitare e forse poco mnemoniche, ma a tutti gli effetti corrette e correttamente gestite.

Grazie al «punto decimale» invece si può ulteriormente favorire la leggibilità di un programma e soprattutto delle variabili: si può chiamare una variabile «coefficiente» se ciò contribuisce alla comprensione del programma, ed ancora meglio si possono usare nomi «doppi» o genericamente «multipli» per le variabili, creando ad esempio «mostri» del tipo:

media.dei.valori
fattore.di.conversione

senz'altro più «pesanti» nello scriverli e nell'occupazione di memoria, ma infinitamente più comprensibili di «sgorbi» quali

mediadeivalori
fattorediconversione

nonché dei «telegrafici»

m
f

oppure

mdf
fdc

che alcuni computer dotati di «dialetti» non accettano perché di lunghezza superiore ai due caratteri.

## Conclusioni

La prossima puntata ci occuperemo, con le solite avvertenze, della routine di calcolo di un'<espressione>, che già sappiamo essere posta in memoria a partire dall'indirizzo 19F3H.

MC

# ...A PROPOSITO DI STAMPANTE, SEI SICURO D'AVER DATO IL MEGLIO AL TUO COMMODORE C 64?

**CARATTERISTICHE TECNICHE**

- Velocità di stampa di 120 caratteri/secondo, bidirezionale e ottimizzata, monodirezionale in near letter quality e grafica.
- Testina di stampa a 9 aghi, larghezza carta 80 colonne.
- Matrice di stampa: 8x11 (standard), 8x6 (grafica a blocchi), 7x60 punti per pollice (grafica bit image), 17x11 (carattere definibile).
- Stampa in normale (10 CPI), condensato (17 CPI), NLQ (10 CPI), espanso (2, 3 o 4 volte), enfatizzato e italico.
- Alimentazione carta a moduli continui (trattore) e fogli singoli (frizione).
- Altre caratteristiche: segnalazione d'errore, caratteri definibili dall'utente, autotest, near letter quality, spoiler per strappo carta, grafica e indirizzamento di singoli punti, dump esadecimale.
- Compatibile con tutti gli Home Commodore: VIC20, C16, C64, C128, Plus 4.


adsGlen Milano

## Probabilmente Star SG10C è la stampante ideale per l'utente avanzato di Commodore

che come te,
vuole scrivere lettere,
stampare tabulati,
tracciare disegni...

il tutto velocemente e affidabilmente.

Totalmente compatibile con il Commodore C64, alfanumerica e grafica, con i suoi 120 caratteri/secondo SG 10C è un punto d'arrivo per i possessori di Commodore C64. Grazie alla possibilità di stampare anche caratteri near letter quality (selezione possibile con la semplice pressione di un tasto), SG 10C può essere asservita al Tuo Commodore come una macchina da scrivere.

**COUPON**

*Per maggiori informazioni ritagliare e spedire a:*
CLAITRON - Via Gallarate 211 - 20125 Milano

Nome ..... Ditta .....
Via ..... Tel. .....
Città ..... CAP .....

MC

Inviaci questo coupon, ti segnaleremo il nome e l'indirizzo di un rivenditore della tua zona che ti potrà mostrare la stampante **Star SG10 C** per il tuo Commodore, ma.....
ATTENZIONE!
Se deciderai di acquistare **Star SG10 C** l'invio del coupon ti darà il diritto ad uno **SCONTO DEL 10%**

**1 ANNO DI GARANZIA**

star
**LA TUA STAMPANTE**

DISTRIBUTORE PER L'ITALIA

CLAITRON SpA

socio assodel

*SEDE e UFF. COMM.: via gallarate 211 - 20151 milano*
*tel. (02) 301.00.81 - 301.00.91 (8 linee ric. aut.)*
*Telex n. 313843 CLAIMI*

RSCG

# Quando il gioco si fa serio

"Multipersonal": un termine che vi giunge nuovo perché è stato appena coniato da Honeywell. Multipersonal è il nuovo X-Superteam, un computer tutto italiano che entra nel mondo dell' "industry standard" perché si avvale dei sistemi operativi Unix™* e Xenix™* che offrono un patrimonio applicativo molto vasto.

Concepito per servire contemporaneamente più utilizzatori intenti a soddisfare esigenze diverse, X-Superteam può interconnettersi per formare una rete locale. Quando il gioco si fa serio, pensate a X-Superteam, un grande computer che sa stare anche sotto il tavolo, se occorre.

X-SUPERTEAM®

IL MULTIPERSONAL

SPONSOR UFFICIALE FORNITORE UFFICIALE
SFIDA ITALIANA AMERICA'S CUP '87

* Unix is a trade-mark of AT & T Laboratories. * Xenix is a trade-mark of Microsoft Inc.
Per informazioni scrivere a Honeywell I.S.I. Sviluppo Commerciale, Via Vida, 11 - 20127 Milano

Conoscere e risolvere insieme.

**Honeywell**

Honeywell Information Systems Italia