Calcolo di espressioni

Seconda parte

Proseguiamo in questa puntata l'analisi del calcolo delle espressioni poste all'interno di un programma scritto in MBASIC: nella prima parte abbiamo visto come l'MBASIC codifica le costanti presenti in una generica espressione ed inoltre abbiamo analizzato la routine «1305H» (è questo il suo indirizzo di partenza, in esadecimale), fondamentale in quanto è lei che analizza il testo e decodifica le costanti trovate, depositandone il valore nell'accumulatore interno, il cosiddetto FAC (Floating ACcumulator, che niente ha a che vedere con l'accumulatore dello Z80 o 8080 che sia).

In questa puntata in particolare analizzeremo altre routine molto importanti, le cui chiamate sono poste all'interno della routine «19F3H» della quale abbiamo già accennato più volte.

La routine «19F3H»: analisi di un'espressione logico-aritmetica

Ecco dunque la routine che effettua il calcolo di un'espressione, ponendone il valore finale nel FAC: ne abbiamo parlato la prima volta quando abbiamo analizzato il comando «LET», senza però approfondirne lo studio, mentre nella scorsa puntata abbiamo parlato del fatto che nel calcolo di un'espressione si deve per forza di cose introdurre il concetto di «ricorsività», la cui presenza è confermata da routine che «chiamano se stesse», come accade infatti nella «nostra» routine.

Osserviamo dunque il listato 1: innanzitutto vediamo che la routine riportata in realtà inizia all'indirizzo «19EFH» e cioè qualche byte prima di 19F3H: come vedremo in seguito, l'indirizzo 19EFH sarà l'entry point per la chiamata ricorsiva alla routine di calcolo dell'espressione.

Iniziamo dunque dall'indirizzo che ora ci interessa maggiormente e cioè da 19F3H: vediamo in quale «stato» si trova una certa routine all'atto della chiamata «CALL 19F3H».

In particolare quello che più ci interessa è che la coppia HL deve puntare al byte a partire dal quale inizia l'espressione da analizzare, come dire che nella nostra analisi del testo di un programma scritto in MBASIC, ad un certo punto con HL punteremo proprio all'inizio di un'espressione, ad esempio subito dopo l'"=" di una assegnazione.

L'istruzione «DEC HL» che troviamo all'indirizzo 19F3H arretra di un posto il puntatore in quanto poi, nel corso del calcolo, verrà chiamata la routine 1305H la quale invece come prima cosa incrementa di un'unità il puntatore in questione, in modo tale da puntare sempre ad un «byte nuovo».

Le successive istruzioni di caricamento del registro D e successivo salvataggio (insieme al registro E) nello stack, non sono di immediata comprensione: in generale possiamo dire che rappresentano lo «stato» dell'analizzatore dell'espressione, che deve essere memorizzato prima di una chiamata ricorsiva (eventuale!) e ripristinato al termine.

Dopo il caricamento ad 1 del registro C troviamo due chiamate a subroutine, la prima alla routine posta all'indirizzo 42BBH e la seconda alla routine posta all'indirizzo 1B84H: dato che queste routine saranno analizzate nei prossimi paragrafi, ora diciamo semplicemente che la prima serve a controllare che lo stack non sia uscito al di fuori della zona a lui concessa, mentre la seconda effettua il vero e proprio calcolo dell'espressione.

Successivamente viene azzerata la locazione 0C09H e salvato il valore del puntatore (la coppia HL) nella locazione 0ABCH: il fatto che nell'istruzione successiva, posta all'indirizzo 1A06H, viene effettuata l'operazione opposta e cioè il caricamento del puntatore HL con il contenuto della locazione 0ABCH, non è evidentemente un errore o un'operazione inutile, in quanto all'indirizzo 1A06H tornano alcune routine innescate durante il calcolo di un'espressione.

La successiva POP BC elimina dallo stack il valore che era stato depositato all'inizio come contenuto della coppia DE, che evidentemente ora non serve più.

Successivamente si ha la lettura del byte puntato da HL, byte del testo che viene analizzato per vedere se l'espressione continua oppure no: in particolare viene effettuato il test se tale byte ha un valore minore di 0EFH, nel qual caso la routine di calcolo dell'espressione viene felicemente abbandonata, per tornare alla parte di programma che aveva effettuato la chiamata.

Invece un valore maggiore o uguale a 0FEH indica la presenza di un operatore logico-algebrico, e perciò si deve proseguire nell'analisi dell'espressione: per ora trascureremo quanto c'è dopo il «RET C», rimandando l'analisi alle prossime puntate.

Invece ora andremo ad analizzare le due subroutine chiamate dalla routine in esame.

La routine 42BBH: il controllo dell'ampiezza dello stack

È questa una routine molto importante, e la sua funzione verrà compresa meglio al termine del paragrafo successivo. Si occupa di controllare che lo Stack Pointer (SP) non sia fuoriuscito dalla zona riservata allo stack e cioè i 512 byte della memoria prima dell'inizio del BDOS (il nucleo fondamentale del CP/M).

Facendo riferimento al listato n. 2, vediamo che la routine in esame è molto semplice in quanto effettua, dopo la sottrazione del valore del massimo indirizzo concesso allo stack dalla costante 0FFC6H, la somma del valore ottenuto con il valore attuale dello Stack Pointer: nel caso in cui tale somma generi riporto (Carry settato), allora tutto va bene e si torna al programma chiamante, altrimenti si effettua (dopo alcune assegnazioni) il salto all'indirizzo 0CE3H con la coppia DE settata al valore 7.

Ciò comporterà l'arresto dell'esecuzione del programma con la visualizzazione del messaggio «Out Of Memory Error in XXXX» dove al posto di XXXX viene indicata la linea che ha generato l'errore di trabocco dello stack.

In effetti arrivati a tale segnalazione d'errore conviene rivedere tutto il programma alla ricerca di espressioni particolarmente onerose, di «FOR» non chiusi da «NEXT», fatti questi che comportano un accrescimento dello stack senza il successivo svuotamento: comunque non è questa la sede adatta all'analisi dettagliata delle cause che possono comportare tale segnalazione d'errore.

Ribadiamo il concetto che in tale sfortunatissimo caso, l'errore o gli errori non saranno quasi mai localizzati nella linea indicata, ma in generale saranno nascosti all'interno del programma: oltre ai FOR ed alle espressioni troppo complicate, altro imputato è il livello di subroutine che si pretende raggiungere, a causa di subroutine che chiamano subroutine, ecc., al di là di livelli di subroutine ottenibili con programmi anche molto complessi.

La routine 1B84H: calcolo di un'espressione

Eccoci dunque alla routine che effettua il vero e proprio calcolo di un'espressione. Osservando il listato n. 3 andiamo subito a vedere la quartultima istruzione di tale routine: è una

Listato 1

```
19EFH   CALL 43C7H
        DEFB '('
19F3H   DEC HL
        LD D,0
19F6H   PUSH DE
        LD C,1
        CALL 42BBH
        CALL 1B84H
        XOR A
        LD (0C09H),A
        LD (0ABCH),A
1A06H   LD HL,(0ABCH)
        POP BC
        LD A,(HL)
        LD (0A90H),HL
        CP 0EFH
        RET C
        ...
```

Disassemblato della routine generale di calcolo di espressioni.

Listato 2

```
42BBH   PUSH HL
        LD HL,(0A76H)
        LD B,0
        ADD HL,BC
        ADD HL,BC
        LD A,0C6H
        SUB L
        LD L,A
        LD A,0FFH
        SBC A,H
        LD H,A
        JP C,4201H
        ADD HL,SP
        POP HL
        RET C
4201H   LD HL,(0796H)
        DEC HL
        DEC HL
        LD (0AB1H),HL
        LD DE,7
        JP 0CE3H
```

Routine di controllo dell'estensione dello stack nel corso del calcolo di espressioni.

CALL 19EFH e cioè una chiamata alla subroutine del listato n. 1, che guardacaso è proprio la routine dalla quale siamo partiti...

Siamo dunque arrivati al punto in cui si innesca un procedimento ricorsivo, gestito, come visto, grazie ad un uso oculato dello stack con il controllo, tramite la già analizzata routine 42BBH, che lo stack si mantenga sempre entro il «livello di guardia». Prima di procedere, ancora una volta desideriamo soffermarci sulla questione della ricorsività con un esempio: supponiamo di voler analizzare l'assegnazione

```
A = INT (SQR (5))
```

tramite la quale associeremo ad A il valore dato dalla parte intera della radice quadrata di 5.

Dal punto di vista dell'MBASIC, ci troveremo all'interno della routine di gestione del comando LET (che in questo caso sappiamo essere implicito, anche se assente) dove, dopo l'assegnazione dell'area alla variabile A ed il controllo dell'effettiva presenza del simbolo dell'"=", viene chiamata la 19F3H per il calcolo di quanto posto dopo l'uguale, seguita poi dall'effettiva scrittura del valore dell'espressione nella zona di memoria riservata alla variabile A.

Entrati nella routine 19F3H, controlleremo il «livello» dello stack dopodiché si passerà alla subroutine 1B84H.

All'interno di questa subroutine, secondo quanto diremo in seguito, salteremo ad 1CC4H e cioè alla gestione delle «funzioni» (nel nostro caso la INT): è qui che sarà presente una nuova chiamata alla 19EFH per il calcolo dell'espressione posta all'interno delle parentesi e su cui deve operare la funzione INT.

Siamo dunque entrati di nuovo nella 19F3H dove, dopo il consueto controllo dello stack, troveremo un'altra chiamata alla 1B84H: qui salteremo ancora ad 1CC4H (gestione delle funzioni) dove ci sarà un'altra chiamata ricorsiva alla 19EFH per il calcolo dell'espressione posta nella parentesi più interna.

Effettuato questo calcolo si ritornerà nella routine di gestione delle funzioni dove troveremo il salto (grazie alla «jump table») alla routine che calcola la radice quadrata di quanto posto nel FAC.

Al ritorno da tale calcolo ci ritroveremo, per un abile gioco di «RET», automaticamente alla routine di gestione delle funzioni, dove ora troveremo il salto (ancora tramite la «jump table») alla routine che effettua il calcolo della parte intera della quantità posta nel FAC. Con un altro RET torniamo alla 19F3H originaria, con la

RET della quale ritorneremo finalmente alla routine di gestione del comando LET.

A questo punto è doverosa una pausa di riflessione.

Entrati all'interno di una 19F3H e passati successivamente nella 1B84H, quest'ultima, come analizzeremo in dettaglio tra poco, memorizzerà nello stack qual è la funzione che desideriamo eseguire e successivamente richiamerà la 19F3H in quanto si aspetterà di trovare una generica espressione come argomento della funzione or ora memorizzata e salvata nello stack.

Procedendo poi a ritroso, una volta calcolato il valore dell'argomento, nello stack ritroveremo proprio l'indicazione di quella che era la funzione da «applicare» all'argomento calcolato.

Finite dunque queste considerazioni andiamo ad analizzare il listato n. 3.

Listato 3

```
1B84H   CALL 1305H      1BC7H   CP 0D6H
        JP Z,0CDEH              JP NZ,1B08H
        JP C,2EC2H              ...
        CALL 44F8H      1B08H   CP 0DCH
        JP NC,1C3BH             JP NZ,1C07H
        CP 20H                  ...
        JP C,135DH      1C07H   CP 0D2H
        INC A                   JP Z,1DCCH
        JP Z,1CC4H              CP 0DAH
        DEC A                   JP Z,49BFH
        CP 0F2H                 CP 0DDH
        JP Z,1B84H              JP Z,4263H
        CP 0F3H                 CP 0D8H
        JP Z,1C2DH              JP Z,48F4H
        CP 22H                  CP 85H
        JP Z,46BCH              JP Z,5513H
        CP 0D5H                 CP 0D3H
        JP Z,1D40H              JP Z,1E45H
        CP 26H                  CALL 19EFH
        JP Z,1C5AH              CALL 43C7H
        CP 0D7H                 DEFB ')'
        JP NZ,1BC7H             RET
        ...
```

Routine di gestione di alcune funzioni che possono comparire in espressioni.

Innanzitutto troviamo una chiamata alla 1305H con la quale analizziamo il byte successivo: segue dunque una serie di test che ci consentono di saltare all'apposita routine in funzione del byte che abbiamo incontrato.

In particolare se il byte letto è nullo oppure pari a 3AH (":"), allora salteremo alla routine di segnalazione di errore di sintassi: ciò capita se tronchiamo un'espressione, mentre la impostiamo, con un «RETURN» oppure con i ":" di separazione con la prossima istruzione.

Nel caso in cui il byte letto sia una cifra espressa in ASCII oppure nel caso in cui sia una lettera dell'alfabeto, allora si salterà alle opportune routine sulle quali non interessa soffermarci ora: in particolare il secondo caso è quello in cui all'interno di un'espressione troviamo una variabile.

Se invece il valore è minore di 20H allora salteremo alla routine 135DH tramite la quale trasferiremo il contenuto di un certo accumulatore temporaneo nel FAC: è questo il caso di cui abbiamo parlato la scorsa puntata, in cui è presente una costante già codificata, da trasferire appunto dalla «linea di programma» agli accumulatori interni.

Se non è vero neanche questo, si testa se il byte in esame vale 0FFH: è questo un byte di prefisso molto importante, indicante appunto la presenza della codifica di una «funzione». In questo caso, come avevamo già accennato in precedenza, si salterà alla routine posta all'indirizzo 1CC4H, sulla quale ritorneremo in dettaglio nel seguito.

Se il byte non vale nemmeno 0FFH, allora verrà confrontato con una serie di valori relativi ad altrettante funzioni logico-algebriche.

Nel caso in cui il byte non corrisponda a nessuna di queste funzioni, allora si ha la fatidica chiamata alla 19EFH e la successiva chiamata alla 43C7H (per testare la presenza della ")" di chiusura dell'argomento) dopodiché si ritorna finalmente al programma chiamante.

Listato 4

```
1CC4H   INC HL          1D11H   CALL C,29E5H
        LD A,(HL)               POP HL
        SUB 81H                 LD DE,1C39H
        CP 7                    PUSH DE
        JP NZ,1CD9H             LD A,1
        ...                     LD (0CD9H),A
1CD9H   LD B,0          1D1AH   LD BC,019FH
        RLCA                    ADD HL,BC
        LD C,A                  LD C,(HL)
        PUSH BC                 INC HL
        CALL 1305H              LD H,(HL)
        LD A,C                  LD L,C
        CP 5                    JP (HL)
        JP NC,1D00H
        ...             1C39H   POP HL
1D00H   CALL 1C25H              RET
        EX (SP),HL
        LD A,L          1C25H   CALL 19EFH
        CP 0CH                  CALL 43C7H
        JP C,1D11H              DEFB ')'
        CP 1BH                  RET
        PUSH HL
```

Routine di gestione delle istruzioni dell'MBASIC.

In particolare all'indirizzo 19EFH troviamo (vedasi il listato n. 1) la chiamata alla 43C7H che testa la presenza di una «parentesi aperta».

Tornando per un istante alle funzioni particolari il cui «token» viene confrontato con il byte letto, troviamo in particolare il «+» ed il «−» applicabili tanto alla somma e alla differenza di due espressioni generiche (operatore binario) quanto ad un'unica espressione (operatore unario); nel primo caso si trattava di espressioni del tipo:

```
<espr.> + <espr.>
<espr.> − <espr.>
```

mentre nel secondo caso si trattava di espressioni del tipo

```
+ <espr.>
− <espr.>
```

Altre funzioni testate nell'ordine sono:

— le **virgolette** ("), nel caso di assegnazioni di stringhe

— l'operatore logico **NOT**, da applicare ad un'espressione logica

— l'«ampersand» («**&**») che consente di impostare valori numerici espressi direttamente in ottale o in esadecimale

— le due funzioni particolari **ERR** ed **ERL**, riguardanti la gestione degli errori di programmazione

— la funzione **VARPTR**, che fornisce l'indirizzo della zona di memoria riservata ad una certa variabile specificata come argomento

— la funzione **USRx** (dove «x» è una cifra compresa tra 0 e 9), che consente l'esecuzione di routine create dall'utente, in linguaggio macchina

— la funzione **INSTR** di ricerca di sottostringhe all'interno di stringhe

— la funzione **INKEY$** che consente l'input da tastiera di valori

— la funzione **STRING$** che permette di creare delle stringhe

— la funzione **INPUT$** anch'essa per l'input da tastiera

— ed infine la funzione **FNX** dove al posto della X c'è una lettera a scelta dell'utente e che identifica una certa funzione creata dall'utente stesso, non a livello basso, ma come insieme di istruzioni MBASIC.

La routine 1CC4H: gestione delle funzioni

Eccoci alla routine, riportata nel listato n. 4, che permette la gestione delle funzioni dell'MBASIC, quando il byte letto dalla routine 1B84H è pari ad FFH. Abbiamo già detto che tale valore rappresenta il prefisso di identificazione delle funzioni, ad ognuna delle quali corrisponde un ben determinato «token», secondo quanto riporta la «jump table» posta a pag. 164 del n. 38 di MC.

In realtà tale tabella in tal senso è errata (ce ne scusiamo con i lettori...) in quanto al token indicato nella colonna di destra e relativo alle istruzioni dell'MBASIC deve essere aggiunto il valore 80H.

È così che ad esempio il token dell'istruzione **SIN** vale **88H** e quello dell'istruzione **MKD$** vale **0B4H**, fermo restando che in questi due casi come in tutti i casi di «istruzioni», il token è preceduto dal prefisso FFH.

Fatta questa doverosa precisazione, ritorniamo alle nostre analisi. Osservando il listato n. 4 vediamo che la routine in esame incrementa innanzitutto HL per poter leggere il «token»

al cui valore sottrae subito 81H, ottenendo così valori compresi tra 00H e 33H. Subito dopo si ha il test se il valore «ridotto» è 7, nel qual caso la funzione considerata è la **RND**: viene trattata come caso a parte in quanto è l'unica che può non essere seguita da un argomento tra parentesi.

Non trattandosi di tale funzione, il valore «ridotto» del token viene ulteriormente raddoppiato, posto nella coppia di registri BC e salvato nello stack (importantissimo! Servirà più tardi).

Un successivo test se il valore «modificato» del token è minore di 5 consente di trattare a parte il caso delle tre istruzioni di stringa, **LEFT$**, **RIGHT$** e **MID$**, le quali hanno la particolarità di richiedere due o tre argomenti tra parentesi, rispettivamente due per la LEFT$ e la RIGHT$, mentre tre argomenti per la MID$. Se la funzione non è nemmeno una di queste tre, allora si ricade nel caso più generale e perciò si arriva all'indirizzo 1D00H dove troviamo una CALL 1C25H: andando a vedere che cosa contiene tale subroutine troviamo, ma oramai non è più una sorpresa, una ennesima CALL 19EFH e cioè la ricerca di una «(» ed il successivo calcolo di un'espressione (ricorsività...), nonché la ricerca alla fine della «)» per pareggiare i conti.

Al ritorno da tale subroutine si trova un'istruzione **EX (SP), HL** tramite la quale carichiamo in HL quanto era posto nella cima dello stack e viceversa depositandovi il valore contenuto in HL: se riflettiamo un istante, nello stack avevamo salvato nientemeno che il token «modificato» dell'istruzione MBASIC che deve agire sull'argomento or ora calcolato...

Ecco che dunque la chiamata ricorsiva alla ben nota 19F3H ha comportato la memorizzazione nel FAC del valore dell'argomento, pronto dunque per essere ulteriormente elaborato.

Prima di fare ciò, però, viene effettuato il test se la funzione in questione non sia per caso compresa fra le seguenti:

SQR, RND, SIN, LOG, EXP, COS, TAN, ATN

nel qual caso il «tipo» del risultato viene forzato, per default, a reale (singola precisione).

Successivamente ed anche in caso che la funzione non era tra quelle citate, viene posto nello stack il valore 1C39H che rappresenterà un indirizzo di ritorno comune, che ritroveremo fra breve.

Infine, in base al valore modificato del token, che ricordiamo essere ora posto in HL, si va a leggere l'oramai famosa «jump table» (il cui indirizzo iniziale è 019FH) ed in particolare si andrà a leggere l'entry point (due byte) della routine che implementa la funzione richiesta: questo indirizzo viene dunque posto in HL e viene effettuato un salto indiretto tramite un'istruzione JP (HL).

In aggiunta al fatto che si è saltato ad una routine (e perciò non si è effettuata una CALL), c'è la considerazione che tutte le routine relative alle varie funzioni terminano con un'istruzione RET; ecco dunque il motivo per cui poco fa era stato salvato nello stack il valore 1C39H: a seguito della RET di chiusura, ogni routine ritorna proprio all'indirizzo 1C39H dove troviamo una POP HL che serve ad estrarre e ripristinare il vecchio valore contenuto in HL, salvato a seguito dell'istruzione EX (SP), HL.

La successiva RET fa ritornare correttamente al programma chiamante per chiudere il ciclo.

Con questo abbiamo terminato: nella prossima puntata proseguiremo con altre informazioni sull'argomento, per poi passare all'analisi delle singole funzioni.

MC