COSTRUIAMOCI UN VERO MICROELABORATORE

HOME COMPUTER AMICO 2000

a cura della A.S.E.L. srl. - parte tredicesima

Si sa che tutti i computers hanno bisogno di istruzioni adeguate per il loro funzionamento e che una serie di istruzioni costituisce un programma.

Un programma deve essere scritto in un determinato modo o linguaggio di programmazione, tale da essere compreso dall'elaboratore.

Esistono essenzialmente due tipi di linguaggi: il primo orientato verso l'elaboratore, il più conosciuto dei quali è l'Assembler che è diverso da computer e computer ed è in stretta relazione con il linguaggio macchina vero e proprio, cioè quello costituito da istruzioni espresse in numeri binari.

Il secondo tipo di linguaggio è quello orientato verso il problema, è più vicino al nostro modo di esprimerci e quindi è più facile da capire e da usare. Di questo secondo gruppo fanno parte ad esempio:

- *ALGOL*
  (Algorithmic Linguage - Linguaggio simbolico)
- *FORTRAN*
  (Formula Translactor)
- *COBOL*
  (Common Business Oriented Language)
- *BASIC*
  (beginners All-purpose Symbolic Instruction Code)
- *PL/M*
  (Program Language for Micriprocessor).

La maggior parte dei programmi scritti in linguaggio orientato verso il problema hanno bisogno di essere tradotti in linguaggio macchina da un ulteriore programma chiamato Compilatore, il quale provvede ad eseguire la traduzione delle istruzioni mnemoniche in istruzioni in codice binario. Una eccezione, ad esempio, è rappresentata dal linguaggio BASIC, che per la sua semplicità è eseguibile dalla macchina tramite un interprete che provvede direttamente, durante l'esecuzione delle istruzioni, alla loro trasformazione in codice binario.

Il Basic come linguaggio è nato intorno agli anni sessanta negli Stati Uniti d'America da un gruppo di ricercatori del Dartmounth College di Hanover nell'Ohio.

In un primo momento questo linguaggio era stato creato per risolvere problemi prettamente scientifici e o didattici. Ma con il passare del tempo, proprio per la sua semplicità e versatilità ebbe un crescente successo e diffusione nella maggior parte delle applicazioni.

L'avvento dei microprocessori e quin-

```
10 PRINT "QUANTI ANNI HAI ?";
20 INPUT A
30 LET M=12
40 LET G=365
50 PRINT " TU HAI ";
60 LET M=A*M
70 PRINT M,"MESI"
80 PRINT "E CON ";A;" ANNI HAI",A*G;" GIORNI"
90 END
```

Fig. 1 - Esempio di semplice programma scritto in BASIC

di dei mini/micro e personal computers associato al Basic ha confermato la validità di questo tipo di linguaggio.

Tanti sono i tipi di macchine che usano la programmazione in Basic, ma dato che ogni tipo di computer possiede diverse caratteristiche si ha come conseguenza l'esistenza di tanti Basic che variano fra di loro per piccole particolarità. Essenzialmente però si può dire che esistono tre tipi di Basic:

mini Basic
standard Basic
Basic esteso

È proprio il primo di questi tre che ci accingiamo a descrivere iniziando per prima cosa a spiegare le varie funzioni ed istruzioni e quindi riportando alcuni esempi esplicativi. Il Basic che prenderemo in esame e che impareremo ad usare si chiama TINY BASIC, cioè "piccolissimo" Basic.

Un programma in Tiny Basic è costituito essenzialmente da istruzioni che sono così composte: numero di riga e istruzione (o istruzioni) vera e propria relativa alla riga.

– *Numero di istruzione.* Ogni riga del programma in Basic deve essere preceduta da un numero non inferiore ad 1 e non superiore a 32.767. Questo numero definisce la giusta sequenza nel programma. Il computer incomincia ad elaborare eseguendo le istruzioni con il numero più basso e prosegue con i successi, fintantoche non incontra un comando che gli dice di agire diversamente, oppure fino a quando non ha raggiunto il numero di riga più elevato.

– *Istruzioni.* Le istruzioni Basic sono parole inglesi che opportunamente inserite in una riga permettono di far eseguire delle determinate funzioni di macchina.

Le istruzioni del Tiny Basic sono:

LET GOTO REM IF.. THEN

GOSUB CLEAR INPUT RETURN

LIST PRINT END RUN

In un programma esiste sempre la necessità di effettuare delle operazioni aritmetiche, e quindi per poter agevolmente risolvere i problemi matematici il Tiny Basic possiede quelle espressioni classiche della aritmetica:

espressione matematica	espressione Basic
Addizione	+
Sottrazione	–
Moltiplicazione	*
Divisione	/

A volte si richiedono particolari tipi di espressioni matematiche che prevedono l'uso delle parentesi ed anche il Tiny Basic prevede per risolvere questo tipo di problema l'applicazione delle parentesi rotonde. Sia in aritmetica che in algebra le espressioni possono essere alquanto complesse: con parentesi tonde, quadre e grafe. In programma scritto in Basic tutte le parentesi si scrivono rotonde. Un esempio potrebbe essere:

{2*[2+(5+4):3]+6}:2

sarà espressa in Tiny Basic:

(2*(2+(5+4)/3)+6)/2

L'interprete Basic eseguirà prima la somma di 5+4 poi dividerà per 3, sommerà 2, moltiplicherà per 2 e sommerà il numero 6 quindi infine dividerà il risultato fin qui ottenuto per 2 ottenendo quale risultato finale 8.

Due ulteriori funzioni di particolare interesse sono previste nel Tiny Basic e si chiamano RND e USR; la prima verrà usata per poter ottenere un numero casuale, mentre la seconda ha la particolarità di poter utilizzare (ovvero richiamare) delle routines definite in linguaggio macchina. Queste funzioni verranno spiegate più ampiamente in seguito.

Come si vedrà, in programmazione si farà largo uso di lettere. Queste lettere inserite in un gruppo di istruzioni prendono il nome di variabili. Per capire meglio che cosa siano queste variabili si potrà dire che esse sono paragonabili a tanti cassetti vuoti che potranno essere riempiti nel corso di un programma. Il programma in Basic userà questi cassetti come accumulatori di dati numerici oppure come termini di paragone od anche per altre funzioni che vedremo in seguito.

Le variabili di cui potremo usufruire con il Tiny Basic sono tante quante sono le lettere dell'alfabeto: A, B, C, D, E.... (cioè 25 variabili).

Analizziamo ora l'uso dei vari gruppi di istruzioni.

Le istruzioni di input/output: LET - INPUT - PRINT

LET. La funzione di LET ci permette di assegnare ad una variabile un determinato valore che potrà essere diretto oppure calcolato. Sarà come se noi mettessimo nel cassetto che avremo chiamato ad esempio A un determinato numero. Allora se noi vogliamo assegnare ad A il numero 12 la nostra istruzione in Basic sarà:

LET A = 12

assegnazione diretta.

Se ad esempio volessimo inserire nel cassetto B un numero corrispondente al totale dei mesi della nostra età (ad esempio 33 anni) scriveremo:

LET B = A * 33

assegnazione calcolata.

Il valore di B sarà uguale a 396 perchè A è uguale a 12 moltiplicato per il numero degli anni (33).

Un altro modo per assegnare un valore diretto ad una variabile sarà quello che in Basic prende il nome di INPUT.

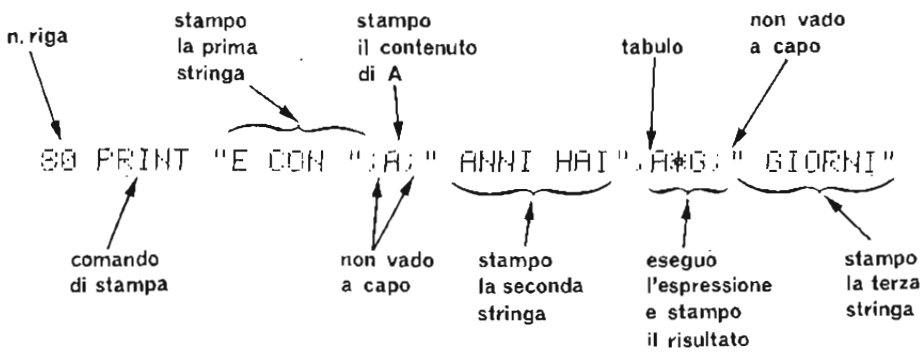

Fig. 2 - Esecuzione di una riga di programma.

INPUT. La istruzione Basic di INPUT possiede, come la LET, la caratteristica di assegnare un determinato valore ad una variabile con la sola differenza sostanziale che per la funzione di INPUT il valore viene introdotto o letto da una periferica che nella fattispecie, nel nostro caso, risulta essere la tastiera.

La forma classica di INPUT è:

```
INPUT A
```

Durante l'esecuzione del programma quando l'interprete trova questo comando abilita la tastiera per ricevere dei valori e non prosegue nella esecuzione delle istruzioni seguenti fintantochè non viene premuto il tasto di RETURN, comando che in questo caso definisce il completamento del dato introdotto. Quindi se noi premiamo in successione i tasti 1 poi il 2 quindi il tasto 3 ed infine il tasto di RETURN, il valore che troveremo in A sarà uguale a 123.

Spesso nei programmi si devono assegnare più valori contemporaneamente a corrispondenti variabili; il Tiny Basic ci permette di ovviare alla necessità di effettuare tanti INPUT quante sono le variabili in questione. L'espressione prima proposta diventa:

```
INPUT A, B, C
```

In tal modo da tastiera si batteranno tutti i valori che dovranno essere assegnati alle variabili previste separandoli con la virgola e premendo quindi ancora il tasto di RETURN per completare l'operazione.

L'interprete riconoscerà il primo valore che termina con la prima virgola, poi il secondo fino alla seconda virgola, e così via fino al completamento dell'istruzione.

Siamo arrivati a questo punto all'ultimo dei comandi di I/O: il PRINT. Questo comando permette di scrivere sullo schermo video un contenuto di variabile oppure una stringa di caratteri od ancora il risultato di una espressione. È importante notare che ciò che si vuole venga "stampato" sul video deve essere messo fra virgolette" ".

Il comando PRINT prevede due sottocomandi: la *virgola* ed il *punto e virgola*.

Il sottocomando *virgola* impone che si esegua una tabulazione sullo schermo e quindi esegue la stampa del contenuto che segue la virgola stessa.

Così ad esempio:

```
PRINT 1, 2, 3, 4
```

sullo schermo apparirà:

```
1    2    3    4
```

Il sottocomando *punto e virgola* ci permette di legare una di seguito all'altro contenuti e/o variabili da stampare;

Con PRINT 1; 2; 3 si otterà sullo schermo 123.

Vediamo con un esempio di programma riportato in *fig. 1* cosa si può fare con le espressioni fin qui imparate. Lo scopo del programma è quello di calcolare il numero dei mesi e dei giorni che ha una persona in base alla sua età (essendo un programma a scopo didattico non si tiene conto degli anni bisestili e del giorno in cui viene posta la domanda all'elaboratore.

Scriviamo il programma *figura 1*.

Vediamo ora cosa è sucesso dopo aver dato il comando di RUN dalla tastiera.

Con la riga 10 è stata eseguita la stampa del contenuto della stringa di caratteri contenuti fra le virgolette. Quindi con il comando *punto e virgola* ci si è fermati sullo schermo sul punto seguente a quello appena stampato, cioè non si è andati a capo alla riga seguente.

Riga 20: abilito la tastiera ed accetto il valore che assegnerò alla variabile A

Riga 30: metto in M il valore 12

Riga 40: metto in G il valore 365

Riga 50: stampo la stringa di caratteri "TU HAI", tabulo e non vado a capo ad una nuova riga

Riga 60: metto in M il valore ottenuto dalla moltiplicazione del contenuto di M precedente che è uguale a 12 moltiplicato per A, il numero degli anni

Riga 70: stampo il nuovo valore di M, tabulo e stampo la parola "Mesi"

Riga 80: in questa riga si riassume in una unica istruzione quelle contenute nelle righe 50, 60 e 70. Vediamo in figura n. 2 che cosa accade durante l'esecuzione di questa riga in programma.

Riga 90: troviamo in questa riga la parola END che a tutti gli effetti è una istruzione in Basic e che deve essere sempre prevista per una fine coretta di ogni programma.

L'istruzione END non deve essere successiva necessariamente alla riga con il valore più alto del programma, ma, come vedremo più avanti, potrà trovare posto in qualsiasi punto del listing (insieme delle istruzioni di un programma), sempre a condizione che tale punto sia previsto e raggiunto alla fine dell'elaborazione.

Nota infine che le istruzioni vengono aumentate di 10 in 10 per convenzione o per comodità nel caso dovessimo inserire altre istruzioni fra due consecutive.

GOTO e GOSUB

Queste due istruzioni Basic vengono definite istruzioni di salto. Molto simili fra di loro, hanno però una particalarità che le distingue. Vediamo per iniziare l'istruzione di:

GOTO. Questa parola di programmazione si può tradurre semplicemente con "Vai alla riga" seguita da un numero corrispondente ad una riga esistente nel programma. L'espressione classica della GOTO è la seguente:

```
    :
    :
 90 LET B = 10
100 GOTO 150
110 LET A = A + B
    :
    :
140 PRINT A
150 PRINT B
    :
    :
```

L'interprete Basic arrivato alla riga 100 riconosce il comando, passa alla riga 150 saltando le righe che vanno da 110 a 140 e quindi esegue l'istruzione che trova in tale riga (in questo caso quella di scrivere sullo schermo il valore della variabile numerica B).

Il Tiny Basic, permette di costruire il numero di riga del GOTO tramite una espressione; allora la riga 100 poteva essere scritta ad esempio in questo modo:

```
100 GOTO 140 + B.
```

GOSUB. Tutte le regole che abbiamo visto per la GOTO valgono anche per la GOTOSUB. Quando noi, diamo il comando di "GOSUB numero-riga" l'interprete del Tiny Basic esegue una memorizzazione del punto in cui viene data questa istruzione, quindi và alla riga di destinazione specificata ed esegue le istruzioni che vengono date da lì in seguito fintanto che non trova un'altra istruzione Basic: *RETURN*, che significa "Ritorno all'indirizzo successivo a quello memorizzato".

Per riassumere il concetto delle istruzioni di GOTO, GOSUB e RETURN si può dire che:

GOTO numero riga = prosegui l'elaborazione della riga segnata o calcola anche se questa nuova riga è inferiore rispetto a quella in cui sei ora. Si può andare quindi anche indietro nelle istruzioni.

GOSUB numero riga = memorizza l'indirizzo dove è stata data questa istruzione, vai nella riga indicata ed esegui i comandi della subroutine che terminerà con la parola RETURN.

RETURN = la subroutine è finita torna indietro all'indirizzo successivo a quello memorizzato.

I F ... THEN ...

In un programma il modo esatto per esprimere questo tipo di istruzione corrisponde a:

IF *operatore di comparazione* THEN *esegui l'istruzione*.

In italiano questa istruzione corrisponde a: SE... ALLORA...

L'operatore di comparazione può essere una variabile oppure una operazio-

ne o anche un valore numerico direttamente espresso.

L'operatore di comparazione stà a significare in che termine si esegue la comparazione vera e propria. Il Tiny Basic prevede essenzialmente sei tipi di operazioni che sono:

termine Basic significato
= uguale a
< minore di
> maggiore di
< = minore o uguale a
= > uguale o maggiore di
< > diverso da

Quando il risultato dell'operazione di comparazione risultasse vera *allora* (THEN) *viene eseguita l'istruzione*, altrimenti l'interprete prosegue alla riga seguente.

Tutte indistintamente le istruzioni Basic possono essere poste dopo la parola THEN.

Per facilitare la comprensione di quanto detto più sopra vediamo alcuni esempi.

1) IF I > 25 THEN PRINT "ERRORE"
2) IF N/P * P = N THEN GOTO 100
3) IF A > B THEN IF B < > C THEN INPUT A, B

Esempio n. 1: *se* la variabile I risulta essere maggiore di 25 *allora* stampa il messaggio posto fra virgolette.
2: *se* la variabile N divisa per la variabile P moltiplicata per se stessa risulta uguale alla stessa variabile N *allora* vai alla riga 100.
3: *se* A è maggiore di B *allora* fai il test: *se* B è diverso da C *allora* chiedi tramite la tastiera i nuovi valori di A e B.

CLEAR. Il termine CLEAR al di fuori di un programma permette la pulizia in memoria di ogni precedente programma. All'interno di un programma cancella il programma stesso. Questo comando è consigliato prima di eseguire il caricamento di un qualsiasi nuovo programma.

END. Come già accennato l'istruzione di END indica all'interprete che l'elaborazione in corso è terminata e che quella è l'ultima istruzione. END deve essere posta come ultima istruzione da eseguire.

REM. Abbreviazione di REMARK, REM ci permette di scrivere qualsiasi commento all'interno del programma. L'interprete riconosce la parola Basic e passa alla istruzione valida seguente.

LIST. Ecco l'istruzione che ci permette di far scorrere sul video il programma residente in memoria. Battendo la parola LIST seguita normalmente dal RETURN si ottiene la stampa su video di tutto il programma oggetto, mentre se si desiderasse vedere una sola linea di istruzioni sarebbe sufficiente impostare "LIST num. riga", RETURN ed ecco apparire la sola riga richiesta. Parimenti se si digitasse LIST num. riga 1, num. riga 2 si otterrebbe la stampa sul video di quella parte di programma delimitata dalle cifre di num. riga 1 e num. riga 2.

Esempi:

LIST	lista tutto il programma
LIST 120	lista la sola riga 120
LIST 75, 150	lista tutte le righe di programma comprese fra la riga 75 e la riga 150

RUN. Questo è il comando che ci permette di far eseguire il programma Tradotto più o meno letteralmente sta a significare "VAI NEL", ed è sottinteso "programma alla prima istruzione". A volte però può essere interessante e necessario eseguire un programma da una determinata istruzione in poi, vuoi per una esecuzione protetta oppure per provare un determinato passo di programma che consente di eseguire un più facile debugging (ricerca dell'errore di programmazione). Per poter far ciò il comando RUN prevede una estensione: accetta cioè il numero di riga di partenza.

Così ad esempio se si impostas RUN 525, l'esecuzione del nostro programma inizia proprio alla riga 525 e poi prosegue come se nulla di diverso alla semplice RUN fosse stato fatto.

Il Tiny Basic prevede come si è già accennato due funzioni intrinseche: RND e USR.

RND. L'interprete del Tiny Basic a questa istruzione genera un numero a caso inferiore al numero posto fra parentesi. L'espressione esatta sarà: RND (numero massimo). Il numero casuale che si ottiene è compreso fra zero ed il numero impostato -1. Se si imposta RND (0) si otterrà, come è logico, un messaggio di errore e conseguente interruzione dell'elaborazione.

USR. La funzione intrinseca del Tiny Basic USR ha come scopo principale quello di eseguire delle routines in linguaggio-macchina. Tre sono le espressioni corrette della istruzione e precisamente:

USR (indirizzo numerico)
USR (indirizzo numerico, registro X)
USR (indirizzonumerico,registo X, registro A).

Quando si definisce indirizzonumericosi intende il normale indirizzo in esadecimale tradotto nel corrispondente valore decimale, così ad esempiol'indirizzo 40AF si traduce in decimale 16459.

Il contenuto del registro-X è il primo oil valore che la subroutine in linguaggio macchina dovrà elaborare e, parimenti il registro A sarà il secondo valore da fornire. Se noi ad esempio in Basic scriviamo:

LET P = USR (12345, 0, 13)

accadrà che l'interprete eseguirà in assembler:

```
LDX   = 0
LDA   = 13
JSR   12345
STA   P
```

troveremo quindi in P il risultato della subroutine in linguaggio macchina sita all'indirizzo decimale 12345 con i valori di 0 e 13 nei rispettivi registri X e A.