FORTH

per VIC 20 e CBM 64

Giacomino Baisini
Gio. Federico Baglioni

GRUPPO
EDITORIALE
JACKSON

FORTH

per VIC 20 e CBM 64

Giacomino Baisini
Gio. Federico Baglioni

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Gli autori ringraziano l'ing. Claudio Fiorentini della G.B.C. Italiana per la costruttiva collaborazione ed interessamento.

Questo libro e' stato scritto e composto con un sistema di Word Processing formato dal computer Commodore VIC 20 e dal WordCraft 20.

Prima edizione: Settembre 1983

PREFAZIONE

I personal computer sono oggi una presenza di rilievo anche nel mercato italiano che puo' contare su decine di migliaia di unita' installate. Ovviamente una grossissima fetta e' costituita da quelle macchine che si possono far rientrare nella definizione di "Home Computer" cioe' computer adoperati in un ambito domestico, familiare.
Ma viene da chiedersi perche' questi apparecchi vengono acquistati e per quali scopi vengano utilizzati. Varie ricerche di mercato hanno evidenziato come in Italia uno dei motivi principali sia quello di apprendere i concetti fondamentali della programmazione. Cio' e' estremamente interessante e inverte la tendenza americana che vuole considerare gli home computer alla stregua di un videogioco evoluto e dotato di tastiera.
Date le premesse e' naturale come in Italia sia aumentata negli ultimi anni la esigenza di libri che trattino tutti gli aspetti dell'informatica ed in particolare della programmazione. Per cio' che riguarda i linguaggi nei quali gli home computer si "esprimono" oggi, la parte del leone viene svolta dal BASIC, di cui sono universalmente note le caratteristiche di semplicita', praticita' ed immediatezza. Ma con l'evolversi delle esigenze dell'utente si pongono all'attenzione altri linguaggi che possono offrire soluzioni molto piu' vantaggiose al programmatore. Il risparmio di energie nella stesura di un programma si traduce spesso in una maggiore eleganza di scrittura ed in una cura per i particolari sovente trascurati in BASIC.
Su queste premesse e' stato sviluppato questo libro sul FORTH, un astro nascente nel firmamento alquanto statico dei linguaggi di programmazione. Penso che i pregi di questa opera siano molteplici primo fra tutti quello di non essere la mera traduzione di un libro americano ma il prodotto dell'impegno dei due giovanissimi autori entrambi studenti della Facolta' di Ingegneria del Politecnico di Milano.
Il secondo pregio e' quello di essere un libro estremamente pratico che fa riferimento ad una implementazione del linguaggio realmente utilizzabile sui computer Commodore VIC 20 e CBM 64. Questa versione del FORTH e' inoltre molto simile a quella disponibile per altri personal computer come ad esempio lo ZX Spectrum.

Il terzo pregio, sempre importante, e' quello di essere il primo testo sul FORTH stampato in italiano, anche se e' auspicabile che se ne possano trovare in breve tempo tantissimi altri.
Non voglio qui dilungarmi sulle caratteristiche del FORTH ne' dare un giudizio sui contenuti del libro lasciando questo compito ai lettori che penso gia' saranno numerosissimi e tutti molto interessati.
Come ultima considerazione vorrei lodare ancora una volta gli autori che, appena ventenni e senza una specifica esperienza, hanno portato a compimento quest'opera in tempi veramente invidiabili. Cio' dimostra come ci siano ancora nelle migliori universita' italiane degli elementi che fanno della serieta' e dell'impegno due regole fondamentali del loro comportamento sia scolastico sia professionale. Grazie a questo rigore i risultati, come si puo' vedere da questo libro, sono conseguiti puntualmente.

Claudio Fiorentini

INDICE

APPENDICI

Capitolo 1

INTRODUZIONE

Nonostante il Basic sia il linguaggio piu' diffuso a livello di personal computer, soprattutto per la sua relativa facilita' di apprendimento e di impiego e per la presenza di una vastissima biblioteca di software disponibile, anche altri linguaggi si stanno diffondendo, grazie alla loro superiorita' rispetto al Basic in specifiche applicazioni.

In questo libro tratteremo del Forth, un nuovo linguaggo di programmazione che, nato per applicazioni su grandi sistemi, puo' dare grandi soddisfazioni anche sui personal computer per le sue innovative caratteristiche e potenzialita'.

La versione cui faremo riferimento e' quella sviluppata dalla Datatronic per il diffusissimo VIC 20 della Commodore, ed e' stata implementata in due memorie ROM da 4 Kbytes situate in un pratico cartridge da inserire nell'apposito alloggiamento sul retro del calcolatore.

Uno degli aspetti che rendono il Forth particolarmente interessante e' l'elevata velocita' di elaborazione, che rende questo linguaggio estremamente adatto all'impiego sia in applicazioni in tempo reale che in programmi molto lunghi e complessi. Per renderci conto di questo basta osservare la tabella di figura 1 nella quale si vede come il calcolatore da noi impiegato, il VIC 20, e' decisamente piu' veloce lavorando in Forth anziche' in Basic. Ad esempio per eseguire 10000 cicli FOR NEXT il calcolatore impiega, in Basic, 9,8 secondi; in Forth, per eseguire altrettanti cicli equivalenti, bastano 1,1 secondi. Altre positive caratteristiche sono la gestione ottimizzata della memoria utente, la presenza di particolari istruzioni per manipolare le stringhe ed una sofisticata gestione dell'INPUT OUTPUT dei floppy-disk.

PROGRAMMI DI TEST IN BASIC

CICLI	ISTRUZIONI	TEMPO
Ciclo FOR NEXT	10 FOR I=1 TO 10000 20 NEXT	9,8 sec.
Addizione intera	10 A=2:B=3 20 FOR I=1 TO 10000 30 C=A+B:NEXT	3,05 sec.
Addizione frazionaria	10 A=3,1416:B=1,4142 20 FOR I=1 TO 1000 30 C=A+B:NEXT	3,15 sec.
Moltiplicazione intera	10 A=2:B=3 20 FOR I=1 TO 1000 30 C=A*B:NEXT	3,88 sec.
Concatenazione di stringhe	10 A$=" ":FOR I=1 TO 255 20 A$=A$+" " 30 NEXT I	1,35 sec.

PROGRAMMI DI TEST IN FORTH

CICLI	ISTRUZIONI	TEMPO
Ciclo DO LOOP	: AAA 10000 1 DO LOOP ;	1,20 sec.
Addizione con numeri semplici	0 VARIABLE A 0 VARIABLE B 0 VARIABLE C : AAA 2 A ! 3 B ! 1000 1 DO A @ B @ + C ! LOOP ;	0,80 sec.
Addizione con numeri doppi	0. 2VARIABLE A 0. 2VARIABLE B 0. 2VARIABLE C : AAA 3.1416 A 2! 2.4142 B 2! 1000 1 DO A 2@ B 2@ D+ C 2! LOOP ;	2,70 sec.

Moltiplicazione con numeri semplici	0 VARIABLE A 0 VARIABLE B 0 VARIABLE C : AAA 2 A ! 3 B ! 1000 1 DO A @ B @ * C ! LOOP ;	1,40 sec.
Concatenazione di stringhe	: AAA 255 1 DO 4500 I + 56 C! LOOP ;	< 0,10 sec.

Fig. 1

Capitolo 2

ORIGINI DEL FORTH

La nascita del Forth risale al 1970 quando un noto programmatore americano, Charles Moore, inizio' a pubblicizzare il suo nuovo linguaggio di programmazione su cui lavorava gia' da alcuni anni: e' infatti noto come l'invenzione e la messa a punto di un linguaggio richiede tempi particolarmente lunghi, soprattutto se innovativo come il Forth.

Considerato da Moore un linguaggio della quarta generazione il Forth prende il nome dall'aggettivo inglese "fourth", che significa appunto quarto. L'abbrevazione da fourth a Forth e' dovuta al fatto che Moore sperimentava il suo nuovo linguaggio su uno dei primi calcolatori interattivi: l'IBM 1130, che era in grado di accettare solo cinque caratteri identificatori.

All'inizio il Forth e' stato impiegato soprattutto dagli astronomi in quanto la sua elevata velocita' di elaborazione lo rende estremamente adatto ad applicazioni e simulazioni in tempo reale. Negli anni successivi il Forth e' stato sempre piu' perfezionato in modo da permetterne l'applicazione e la diffusione anche in altri campi dell'informatica.

Si stima che nel 79 i programmatori di Forth fossero, negli USA, circa 1000, e siano oggi oltre 10000. Solo recentemente si sono pero' rese disponibili le prime implementazioni del Forth non solo sui grandi sistemi ma anche sui personal computer piu' diffusi.

Capitolo 3

DEFINIZIONE DI NUOVE PAROLE

La prima istruzione che possiamo utilizzare del Forth e' VLIST. Compito di questa istruzione e' visualizzare sul video l'intero vocabolario dell'interprete Forth, vocabolario che puo' essere utilmente ampliato dal programmatore.

Digitiamo dunque VLIST seguito da <RETURN>. Sullo schermo televisivo scorreranno le parole chiave del linguaggio, scorrimento che potra' essere rallentato tramite il tasto <CTRL> e bloccato con il tasto <RUN/STOP>.
Dopo aver visto il vocabolario ripuliamo lo schermo tramite l'usuale tasto <CLR> e impariamo ad usare l'istruzione ." " che serve a visualizzare una serie di caratteri alfanumerici racchiusa tra i due apici.

Esempio:

Digitando:

." CIAO " <RETURN>

sullo schermo apparira' la scritta:

CIAO OK

Si noti come le parole-chiave siano separate da uno spazio, che permette all'interprete Forth di distinguere le varie istruzioni, e che alla fine di ogni elaborazione compaia sullo schermo la scritta OK.

Vediamo ora, tramite alcuni esempi, come ampliare il vocabolario. La procedura da seguire per definire una nuova parola e' la seguente:

si digita

: nome-istruzione definizione ;

Esempio:

: A1 ." IL MARE " ; <RETURN>

Con l'istruzione <:> predisponiamo l'interprete Forth ad accettare una nuova parola; A1, che digitiamo preceduto e seguito da uno spazio, e' il nome che abbiamo assegnato alla nuova parola. La parte rimanente dell'istruzione rappresenta cio' che deve essere eseguito dal calcolatore. L'istruzione <;> finale fa ritornare il calcolatore nel modo operativo normale.
In questo modo abbiamo creato la nuova parola A1 che, ogni volta che verra' impiegata, fara' comparire sullo schermo la scritta IL MARE. Infatti digitando:

A1 <RETURN>

apparira'

IL MARE OK

Per verificare che la nuova parola A1 e' parte integrante del vocabolario dell'interprete Forth, impieghiamo ora l'istruzione VLIST gia' precedentemente incontrata.

Digitiamo dunque:

VLIST <RETURN>

apparira'

A1 CLOAD CSAVE

B. H. DUMP

K J I' UM*

ecc.

ecc.

confermando come la parola A1 sia stata assunta dall'interprete Forth.
Continuando con gli esempi definiamo altre nuove parole seguendo la prassi che abbiamo appena imparato.

Digitiamo dunque:

: A2 ." LA NAVE " ; <RETURN>

: A3 ." BAGNA " ; <RETURN>

: A4 ." ATTRAVERSA " ; <RETURN>

Premendo nuovamente VLIST il calcolatore visualizzera':

A4 A3 A2 A1

CLOAD CSAVE

B. H. DUMP

K J I' UM*

ecc.

confermando come pure queste altre nuove parole siano state assunte dall'interprete nel proprio vocabolario.
A questo punto definiamo un'altra parola cosi' strutturata:

: B1 A2 A4 A1 ; <RETURN>

Se ora digitiamo B1 apparira' sul video:

LA NAVE ATTRAVERSA IL MARE OK

E' interessante notare come la nuova parola B1 sia stata definita in base alle tre parole A2 A4 A1 che originariamente non erano presenti nell'interprete Forth.
Continuando nella creazione di nuove parole, definiamo pure B2:

: B2 A1 A3 A2 ; <RETURN>

Sul video apparira' la scritta:

IL MARE BAGNA LA NAVE OK

I semplici esempi appena visti ci mostrano come sono strutturati i programmi in Forth: non esiste un limite alla concatenazione delle parole ed inoltre una stessa parola (vedi A1 A2 A3 A4) puo' essere utilizzata in piu' di un programma.

Tornando ai nostri esempi precedenti, nel caso si siano commessi degli errori nell'inserimento delle istruzioni, potremo utilmente impiegare la parola-chiave FORGET.
Ad esempio, se volessimo cancellare l'ultima parola definita, B2, dovremmo operare in questo modo:

FORGET B2 <RETURN>

Fatto questo, la parola B2 verra' tolta dal vocabolario. Bisogna pero' tener presente che l'istruzione FORGET cancella dal vocabolario, oltre all'istruzione desiderata, anche tutte quelle che sono state immesse dopo. Infatti, eseguendo FORGET A3, anche le parole A4 e B1 verranno cancellate, in quanto definite posteriormente ad A3. Un VLIST ci confermera' tutto questo:

VLIST <RETURN>

appariranno:

A2 A1 CLOAD

CSAVE B. H.

DUMP K J I'

ecc.

ecc.

Prima di procedere oltre resettiamo il sistema con l'istruzione COLD che reinizzializza il calcolatore riportandolo alle condizioni seguenti l'accensione.

REGISTRAZIONE VOCABOLARIO

Se, una volta introdotte delle nuove parole-chiave, vogliamo conservarle anche dopo lo spegnimento del calcolatore, possiamo registrarne il nome e la definizione su cassetta impiegando il comando CSAVE.
Digitato il comando CSAVE sul video apparira' la scritta:

PRESS RECORD & PLAY ON TAPE

e, premuti contemporaneamente i due tasti del registratore, le scritte:

OK

SAVING

confermeranno l'inizio della registrazione. Alla fine della registrazione stessa apparira' infine la scritta:

OK

Per ricaricare in memoria delle parole memorizzate su cassetta va invece impiegato il comando CLOAD, posizionando prima esattamente il nastro.
Il calcolatore rispondera':

PRESS PLAY ON TAPE

e la fine del caricamento sara' confermata dalla scritta:

OK

Nel caso si siano verificati degli errori in fase di caricamento o di lettura, nella posizione P1 dello Stack sara' presente il valore 1 e apparira' la scritta:

? LOAD ERROR

Se le operazioni di caricamento o lettura si sono svolte invece correttamente, nella posizione P1 dello Stack sara' presente il valore 0.

Volendo associare ad una registrazione un nome, si impiega l'istruzione EXPECT nel seguente modo:

addr n EXPECT <RETURN> nome <RETURN>

dove

addr indica l'indirizzo in cui memorizzare il nome

n indica la lunghezza del nome

nome e' nome assegnato alla registrazione

Abbiamo cosi' memorizzato in una particolare zona della memoria un nome, che potra' essere associato alla registrazione nel seguente modo:

addr n NAME CSAVE <RETURN>

e, per la lettura:

addr n NAME CLOAD <RETURN>

Esempio:

Volendo registrare la definizione delle parole introdotte associando loro il nome LUNA, digiteremo:

1024 4 EXPECT <RETURN> LUNA <RETURN>

1024 4 NAME CSAVE <RETURN>

Potremo leggere la registrazione di nome LUNA digitando:

1024 4 EXPECT <RETURN> LUNA <RETURN>

1024 4 NAME CLOAD <RETURN>

Capitolo 4

FUNZIONAMENTO DELLO STACK

Per poter proficuamente lavorare con il Forth e' necessario aver ben chiaro il concetto di Stack in quanto, a differenza di altri linguaggi, nel Forth essa e' facilmente accessibile al programmatore.
Lo Stack e' una zona della memoria, che inizia alla locazione 4 e termina a 116, destinato a contenere i numeri che vengono utilizzati nei calcoli.
Per il suo funzionamento potrebbe essere paragonato ad una pila di piatti in cui ogni piatto corrisponde ad un numero: immaginiamo ogni numero che immettiamo nel calcolatore come l'aggiunta di un piatto alla pila. Dunque, se alla pila aggiungiamo piatti, questa si alza (e, per analogia, lo Stack si riempie di numeri). Immaginiamoci ancora per un attimo di fronte a questa pila di piatti con la necessita' non di aggiungerne, ma di prenderne uno. Per forza di cose siamo costretti a prendere l'ultimo alla sommita': analogamente, se richiediamo un numero allo Stack, questo ci fornira' l'ultimo che abbiamo introdotto.

Per meglio chiarire il concetto facciamo un esempio: scrivendo

1 2 3 4 5 6 <RETURN>

i numeri verranno memorizzati nello Stack uno di seguito all'altro.
Per verificarlo leggiamone dunque il contenuto tramite l'istruzione dot (che si ottiene premendo semplicemente il tasto (.)): digitando per sei volte i tasti (.) e <RETURN>, verranno visualizzati i numeri 6 5 4 3 2 1 in ordine inverso rispetto all'immissione.

Nelle figure che seguono viene data una rappresentazione grafica dello Stack e delle operazioni eseguite su di esso: ogni quadratino, indicato con p1 p2 p3 ..., rappresenta una posizione dello Stack.

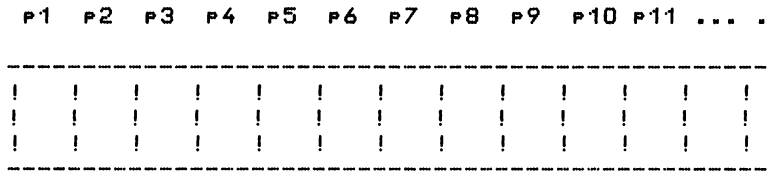


fig. 2

Dalla figura 2 vediamo come lo Stack sia vuota.

tasti da digitare	video	P1	P2	P3	P4	P5	P6	...
1 <RETURN>								
	OK	1						
2 <RETURN>								
	OK	2	1					
3 <RETURN>								
	OK	3	2	1				
4 <RETURN>								
	OK	4	3	2	1			
5 <RETURN>								
	OK	5	4	3	2	1		
6 <RETURN>								
	OK	6	5	4	3	2	1	

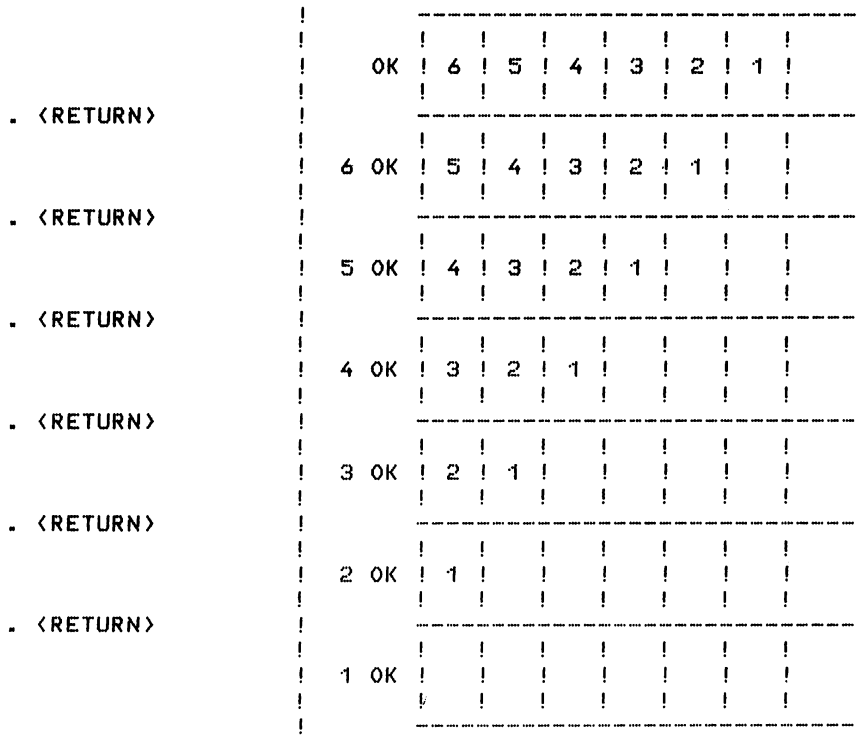


Fig. 3

La figura 3 e' la rappresentazione sequenziale delle operazioni fatte fino ad ora: a sinistra sono indicate le istruzioni digitate, al centro cio' che appare sul video e a destra il conseguente contenuto dello Stack.

E' interessante notare come i numeri, man mano che vengono immessi, occupano inizialmente la posizione P1 spostando verso destra quelli gia' presenti nello Stack: all'inizio lo Stack e' vuoto ed immettendo il numero 1 esso va ad occupare la posizione P1. Con l'introduzione del numero 2, l'1 passera' nella posizione P2 ed il 2 sara' memorizzato in P1.

Analogamente avverra' con i numeri 3 4 5 6 giungendo cosi' alla situazione prospettata in figura 4 dove vediamo che in P1 abbiamo l'ultimo numero immesso (il 6), mentre nella posizione piu' a destra, il primo (l'1).

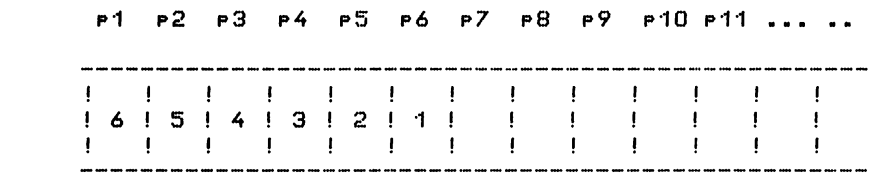


Fig. 4

Inversamente a quanto appena visto, se viene richiesto un numero allo Stack tramite l'istruzione dot, esso fornira' quello contenuto in P1, e cioe' l'ultimo immesso, spostando gli altri di una posizione verso sinistra.

Capitolo 5

PARTICOLARI ISTRUZIONI LEGATE ALLO STACK

Parliamo ora di alcune importanti istruzioni che permettono di modificare la posizione dei numeri nello Stack e di agire su di essi. Ove necessario verra' data la rappresentazione del contenuto dello Stack prima e dopo l'uso dell'istruzione.

. (dot) toglie dallo Stack il valore contenuto in P1 e lo visualizza sullo schermo. Sposta poi tutti gli altri valori di una posizione verso sinistra;

DROP (scarica) analogamente all'istruzione precedente toglie dallo Stack il valore contenuto in P1 senza pero' visualizzarlo;

DUP (duplica) legge il contenuto di P1, sposta tutti i valori contenuti nello Stack di una posizione verso destra, e reimmette quanto letto in P1.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	...	.
7	2	5	9	4								

Fig. 5

Dalla situazione rappresentata in figura 5, si passa, con il DUP a quella di figura 6.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	...	.
7	7	2	5	9	4							

Fig. 6

OVER (sopra) legge il contenuto di P2, sposta tutti i valori contenuti nello Stack di una posizione verso destra e reimmette quanto letto in P1.

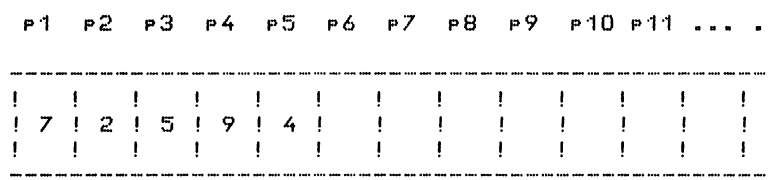


fig. 7

Rappresentazione dello Stack prima (figura 7) e dopo (figura 8) l'uso dell'istruzione OVER.

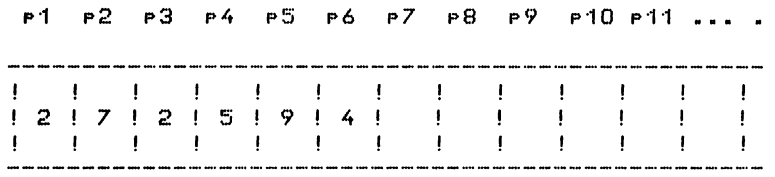


Fig. 8

SWAP (scambia) scambia tra loro i contenuti di P1 e P2

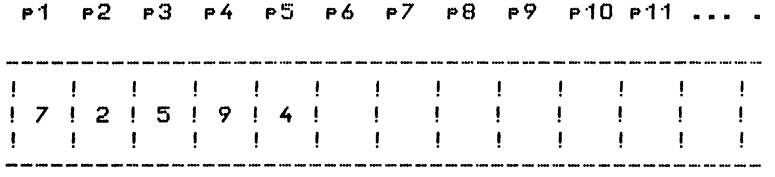


Fig. 9

Rappresentazione dello Stack prima (figura 9) e dopo (figura 10) l'uso dell'istruzione SWAP.

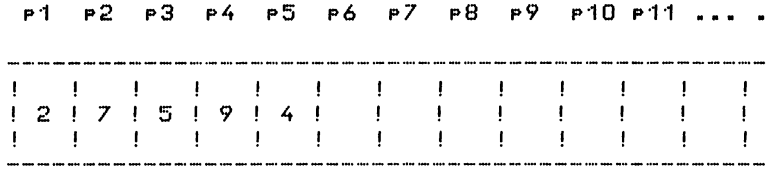


Fig. 10

ROT (ruota) trasferisce il valore contenuto in P3 in P1 e sposta i valori che erano contenuti in P1 e P2 rispettivamente in P2 e P3.

Fig. 11

Rappresentazione dello Stack prima (figura 11) e dopo (figura 12) l'uso dell'istruzione ROT.

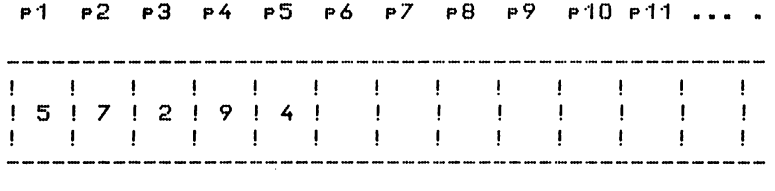


Fig. 12

2DROP (scarica) toglie dallo Stack il numero doppio contenuto in P1 P2. Sposta poi tutti gli altri valori di due posizioni verso sinistra;

2DUP (duplica) legge il contenuto di P1 e P2, sposta tutti i valori contenuti nello Stack di due posizioni verso destra, e reimmette quanto letto in P1 e P2.

```
 P1  P2  P3  P4  P5  P6  P7  P8  P9  P10 P11 ... .
-------------------------------------------------------
!   !   !   !   !   !   !   !   !   !   !   !   !   !
! 7 ! 2 ! 5 ! 9 ! 4 !   !   !   !   !   !   !   !   !
!   !   !   !   !   !   !   !   !   !   !   !   !   !
-------------------------------------------------------
```

Fig. 13

Rappresentazione dello Stack prima (figura 13) e dopo (figura 14) l'uso dell'istruzione 2DUP.

```
 P1  P2  P3  P4  P5  P6  P7  P8  P9  P10 P11 ... .
-------------------------------------------------------
!   !   !   !   !   !   !   !   !   !   !   !   !   !
! 7 ! 2 ! 7 ! 2 ! 5 ! 9 ! 4 !   !   !   !   !   !   !
!   !   !   !   !   !   !   !   !   !   !   !   !   !
-------------------------------------------------------
```

Fig. 14

2OVER (sopra) legge il contenuto di P3 e P4, sposta tutti i valori contenuti nello Stack di due posizioni verso destra e reimmette quanto letto in P1 e P2.

```
 P1  P2  P3  P4  P5  P6  P7  P8  P9  P10 P11 ... .
-------------------------------------------------------
!   !   !   !   !   !   !   !   !   !   !   !   !   !
! 7 ! 2 ! 5 ! 9 ! 4 !   !   !   !   !   !   !   !   !
!   !   !   !   !   !   !   !   !   !   !   !   !   !
-------------------------------------------------------
```

Fig. 15

Rappresentazione dello Stack prima (figura 15) e dopo (figura 16) l'uso dell'istruzione 2OVER.

Fig. 16

2SWAP (scambia) scambia i contenuti di P1 e P2 con quelli di P3 e P4 e viceversa.

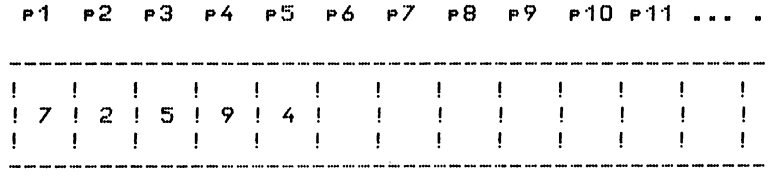


Fig. 17

Rappresentazione dello Stack prima (figura 17) e dopo (figura 18) l'uso dell'istruzione 2SWAP.

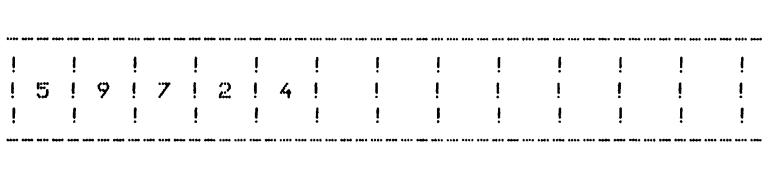


Fig. 18

2ROT (ruota) legge i valori contenuti in P5 e P6, sposta i valori contenuti in P1 P2 P3 e P4 di due posizioni verso destra e reimmette quanto letto in P1 e P2.

Fig. 19

Rappresentazione della Stack prima (figura 19) e dopo (figura 20) l'uso dell'istruzione 2ROT.

Fig. 20

Capitolo 6

MEMORIZZAZIONE DEI NUMERI

Nel Forth e' possibile operare sui numeri in cinque diversi modi a seconda della precisione di calcolo necessaria. Infatti si possono impiegare numeri a singola e doppia precisione, rispettivamente con e senza segno, e numeri a tripla precisione senza segno, la cui differenza consiste nel numero di bytes utilizzati per la memorizzazione.

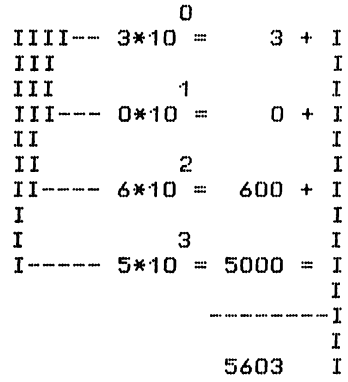
Come ben sappiamo i calcolatori non operano su numeri in base dieci ma bensi' su numeri binari. Se infatti vogliamo introdurre nella memoria un numero decimale, ad esempio 189, esso verra' memorizzato dopo essere stato convertito nell'equivalente binario 10111101.
Volendo conoscere l'equivalente binario di un numero in base dieci abbiamo a disposizione, nel vocabolario Forth, l'apposita istruzione B. utilizzabile per i numeri compresi tra -32768 e +32767. Se infatti carichiamo nello Stack il numero 189 e poi lo estraiamo con l'istruzione B. otteniamo il valore 10111101.
Per poter capire le differenze e l'impiego dei cinque diversi tipi di numeri, e' necessario aprire ora una piccola parentesi sull'aritmetica binaria.

Esaminiamo un numero in base dieci come ad esempio 5603: ogni cifra ha un valore particolare a seconda della sua posizione nel numero.

```
5603


IIII-- unita'
III
III--- decine
II
II---- centinaia
I
I----- migliaia
```

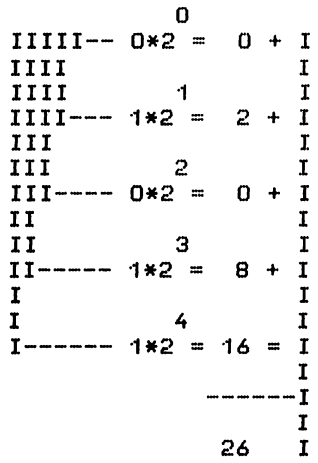
In sostanza il numero 5603 potrebbe essere scritto come:

5603



Abbiamo cioe' moltiplicato ogni cifra del numero per una potenza crescente di dieci.
Se vogliamo operare con una base diversa da quella decimale, useremo le stesse convenzioni gia' impiegate: ad esempio, lavorando con il sistema binario (che usa come base 2 e come cifre 0 e 1), avremo una rappresentazione analoga alla precedente.
Il numero binario 11010 (equivalente a 26 in base decimale) puo' essere scritto:

11010



Notiamo come in quest'ultimo disegno, il numero venga scomposto in cifre moltiplicate per delle potenze crescenti di due anziche' di dieci.

Vediamo ora come calcolare l'equivalente decimale di un numero binario composto solo dalle cifre 1. La formula di conversione e':

$$Ndec. = 2^{Nbc} - 1$$

ove:

Ndec. : numero decimale
Nbc : numero cifre del numero binario

Ad esempio il numero binario 111 equivale a:

$$2^3 - 1 = 8 - 1 = 7$$

Infatti, impiegando sempre lo schema delle tabelle precedenti, si ha:

111

```
          0
III-- 1*2 = 1 + I
II                I
II        1       I
II--- 1*2 = 2 + I
I                 I
I         2       I
I---- 1*2 = 4 = I
                  I
              ----I
                  I
               7  I
```

Nel calcolatore un numero binario viene memorizzato utilizzando un bit per ogni cifra: da quanto visto precedentemente potremo dunque concludere che un byte puo' contenere al massimo il numero binario 11111111 che, convertito in base dieci utilizzando la formula appena vista, corrisponde a:

$$2^8 - 1 = 256 - 1 = 255$$

Esaminiamo ora dettagliatamente i vari modi utilizzati dall'interprete Forth per memorizzare i numeri.

NUMERI SEMPLICI CON IL SEGNO

Vengono memorizzati utilizzando due bytes, ma uno dei 16 bits disponibili e' impiegato per il segno. I bits per la memorizzazione del valore del numero sono percio' 15 ed essendo:

$$2^{15} - 1 = 32767$$

il numero deve essere compreso tra -32768 e +32767. In tutti gli esempi fatti fino ad ora sono stati utilizzati numeri semplici con il segno che, come abbiamo visto, possono essere caricati nello Stack semplicemente con il tasto <RETURN>, e visualizzati sullo schermo con l'istruzione dot (ottenuta con il tasto <.>).

NUMERI SEMPLICI SENZA SEGNO

Vengono memorizzati utilizzando 16 bits ovvero due bytes. Il valore massimo che possono assumere questi numeri e' quindi:

$$2^{16} - 1 = 65535$$

Vanno immessi nello Stack digitando il tasto <RETURN> dopo il numero.
Per visualizzare sullo schermo un numero semplice senza segno contenuto nella posizione P1 dello Stack, si ricorre all'istruzione U. (abbreviazione di unsigned = senza segno).

NUMERI DOPPI CON IL SEGNO

Vengono memorizzati utilizzando 4 bytes ma uno dei 32 bits disponibili e' utilizzato per il segno.

I 31 bits restanti permettono quindi l'utilizzo di valori compresi tra -2.147.483.648 e +2.147.483.647. Infatti:

$$2^{31} - 1 = 2.147.483.647$$

I numeri doppi possono essere caricati nello Stack semplicemente interponendo tra le loro cifre, in una qualsiasi posizione, il punto decimale e premendo il tasto <RETURN>.
Ad esempio il numero 257 puo' essere caricato nello Stack come numero doppio, indifferentemente in uno dei seguenti modi:

257. <RETURN>

25.7 <RETURN>

2.57 <RETURN>

.257 <RETURN>

A differenza dei numeri semplici, l'interprete Forth assegna ai numeri doppi non una ma due posizioni nello Stack. Questo e' interessante perche', in realta', un numero doppio viene in sostanza considerato come formato da due numeri semplici.
L'interprete Forth carica nello Stack i numeri doppi spezzandoli in due numeri semplici. Esempio:

Caricando il numero 86212 come numero doppio, cioe' digitando:

86212. <RETURN>

verra' in realta' immesso, nella posizione P1 dello Stack il numero semplice con segno 1, mentre nella posizione P2 il numero semplice senza segno 20676. L'operazione compiuta dall'interprete per ottenere questi due numeri e' la seguente:

1) divide il numero doppio per 65536 (2 16);
2) carica il quoziente in P2;
3) carica il resto in P1.

Per richiamare un numero doppio contenuto nello Stack si impiega l'istruzione D. seguita da <RETURN>.

NUMERI DOPPI SENZA SEGNO

Vengono memorizzati utilizzando 4 bytes ovvero 32 bits. Il valore massimo che possono assumere e' dunque:

$$2^{32} - 1 = 4.294.967.295$$

Vanno immessi nello Stack con la stessa procedura utilizzata per i numeri doppi con segno. E' possibile eseguire delle operazioni con questi numeri ma non esistono delle istruzioni specifiche per toglierli dallo Stack.

NUMERI TRIPLI SENZA SEGNO

Vengono memorizzati utilizzando 6 bytes ovvero 48 bits. Il valore massimo che possono assumere e' dunque:

$$2^{48} - 1 = 2{,}814E14$$

Pure con i numeri tripli e' possibile eseguire delle operazioni, ma non esistono specifiche istruzioni per immetterli o toglierli dallo Stack.

A conclusione di questo capitolo ci sembra importante riesaminare il concetto di Stack: esso e' costituito semplicemente da un insieme di bytes i cui contenuti possono essere scambiati tra loro utilizzando le istruzioni viste nel capitolo precedente. Lo Stack e' formato da 112 bytes ed ognuna delle sue posizioni e' costituita da due bytes.

Tutte le istruzioni che abbiamo finora incontrato sono semplicemente dei modi codificati per agire sul contenuto dei bytes dello Stack indipendentemente dal loro contenuto. Per esempio possiamo caricare due numeri semplici e leggerli come un singolo numero doppio o operare uno SWAP sui due numeri che costituiscono un numero doppio ottenendo cosi' un valore completamente diverso da quello originario.

Capitolo 7

OPERAZIONI MATEMATICHE

Come abbiamo visto nei capitoli precedenti, ogni valore immesso nel calcolatore viene memorizzato nello Stack sul quale e' possibile agire con le varie istruzioni fin'ora incontrate. In questo capitolo parleremo di altre importanti istruzioni agenti sui numeri contenuti nello Stack: le operazioni aritmetiche, relazionali e logiche.

E' pero' prima necessario sapere che l'interprete Forth opera in un particolare tipo di notazione matematica, detta RPN (reverse Polish notation), che presenta innumerevoli vantaggi ed e' impiegata in parecchi sistemi elettronici quali, ad esempio, i compilatori Pascal e le calcolatrici della Hewlett Packard. Dall'esempio seguente possiamo dunque osservare come operare in RPN eseguendo una somma tra due numeri.

La sequenza di operazioni da compiere e' la seguente:

tasti da digitare	video	contenuto dello Stack
5 RETURN	OK	5
7 RETURN	OK	7 5
+ RETURN	OK	12
. RETURN	12 OK	

Fig. 21

Si noti che si e' scritto

5 7 +

e non

5 + 7

in quanto l'interprete opera, come gia' detto, in RPN. Alcuni vantaggi presentati da questo tipo di notazione matematica sono la maggior velocita' di immissione dati e la possibilita' di scrivere ad esempio un'equazione immettendo prima tutti i numeri e poi tutti i segni operazionali.

Ad esempio la semplice equazione:

4-2* 7+5*(2+6)

andra' immessa come:

4 2 7 5 2 6 + * + * -

In sostanza, ricordando la struttura dello Stack, in RPN si opera immettendo prima i numeri e poi i segni operazionali tenendo conto che essi agiscono sempre sul contenuto delle posizioni P1 e P2 dello Stack.

OPERAZIONI ARITMETICHE

Parliamo dunque delle operazioni aritmetiche disponibili e del loro impiego a seconda del tipo di numeri impiegati.

Numeri semplici

Le operazioni matemetiche eseguibili con i numeri semplici che, come ricordiamo, vengono memorizzati occupando due bytes, sono le seguenti:

+ (somma) esegue la somma dei due numeri contenuti in P2 e P1 mettendo il risultato cosi' ottenuto in P1 e spostando tutti i valori contenuti nello Stack di una posizione verso sinistra.

- (differenza) esegue la differenza tra i due numeri contenuti in P2 e P1 mettendo il risultato cosi' ottenuto in P1 e spostando tutti i valori contenuti nello Stack di una posizione verso sinistra.

`*` (prodotto) esegue il prodotto tra i due numeri contenuti in P2 e P1 mettendo il risultato cosi' ottenuto in P1 e spostando tutti i valori contenuti nello Stack di una posizione verso sinistra.

`/` (rapporto) esegue il rapporto tra i due numeri contenuti in P2 e P1 mettendo il risultato cosi' ottenuto in P1 e spostando tutti i valori contenuti nello Stack di una posizione verso sinistra.

`/MOD` (rapporto) esegue il rapporto tra i due numeri contenuti in P2 e P1. Il quoziente verra' memorizzato in P1 ed il resto in P2.

`*/` esegue il prodotto tra i numeri contenuti nelle posizioni P3 e P2 dello Stack, divide il risultato per il valore di P1 mettendo il quoziente in P1 e sposta tutti i valori contenuti nello Stack di due posizioni verso sinistra.

`*/MOD` esegue il prodotto tra i numeri contenuti nelle posizioni P3 e P2 dello Stack, divide il risultato per il valore di P1 mettendo il quoziente in P1 ed il resto in P2; sposta infine tutti i valori contenuti nello Stack di una posizione verso sinistra.

`MOD` esegue il rapporto tra P2 e P1 scrivendo il resto in P1 e sposta il contenuto dello Stack di una posizione verso sinistra.

`+-` legge il valore di P1, gli attribuisce il segno di P2 e, dopo aver spostato il contenuto dello Stack di una posizione verso sinistra, scrive il risultato in P1.

`ABS` Fornisce in P1 il valore assoluto del numero ivi precedentemente contenuto.

`MINUS` cambia il segno al valore contenuto in P1.

Numeri doppi

Le operazioni matemetiche eseguibili con i numeri doppi che, come ricordiamo, vengono memorizzati occupando quattro bytes, sono le seguenti:

D+ (somma) esegue la somma dei due numeri doppi contenuti in P1 P2 e P3 P4, mettendo il risultato cosi' ottenuto in P1 P2 e spostando tutti i valori contenuti nello Stack di due posizioni verso sinistra.

D+- legge il numero doppio contenuto in P1 P2, gli attribuisce il segno di P3 e, dopo aver spostato il contenuto dello Stack di due posizioni verso destra, scrive il risultato in P1 P2.

DABS Fornisce in P1 P2 il valore assoluto del numero doppio ivi precedentemente contenuto.

DMINUS cambia il segno al valore contenuto in P1 P2.

Operazioni miste

Con "operazioni miste" si intendono le operazioni eseguite tra numeri di diversa precisione. Esse sono:

M* esegue il prodotto tra i due numeri semplici con segno contenuti in P1 e P2 ottenendo un numero doppio con segno che viene memorizzato in P1 P2.

M/ esegue il rapporto tra il numero doppio con segno contenuto in P2 P3 ed il numero semplice con segno contenuto in P1, memorizzando il quoziente ed il resto rispettivamente in P1 e P2 come numeri semplici con segno; infine sposta il contenuto dello Stack di una posizione verso sinistra.

M*/ Moltiplica il numero doppio con segno contenuto in P3 P4 con il numero semplice con segno contenuto in P2, ottenendo un numero triplo che viene a sua volta diviso per il numero semplice senza segno contenuto in P1. Il risultato e' un numero doppio con segno che viene caricato in P1 P2. Infine viene spostato il contenuto dello Stack di due posizioni verso sinistra.

M/MOD divide il numero doppio senza segno contenuto in P2 P3 con il numero semplice senza segno contenuto in P1. Il quoziente viene memorizzato in P1 P2 come numero doppio senza segno mentre il resto e' un numero semplice senza segno che viene caricato in P3.

U* moltiplica i due numeri semplici senza segno contenuti in P1 e P2 memorizzando il prodotto in P1 P2 come numero doppio senza segno.

U/ esegue il rapporto tra il numero doppio senza segno contenuto in P2 P3 ed il numero semplice senza segno contenuto in P1 memorizzando il quoziente ed il resto come numeri semplici senza segno rispettivamente in P1 e P2. Infine sposta il contenuto dello Stack di una posizione verso sinistra.

UM*/ Moltiplica il numero doppio senza segno contenuto in P3 P4 con il numero semplice senza segno contenuto in P2 ottenendo un numero triplo che divide per il numero semplice senza segno contenuto in P1. Il risultato e' un numero doppio senza segno che viene caricato in P1 P2. Infine viene spostato il contenuto dello Stack di due posizioni verso sinistra.

T* moltiplica il numero doppio senza segno contenuto in P2 P3 con il numero semplice senza segno contenuto in P1; il numero triplo senza segno cosi' ottenuto viene memorizzato in P1 P2 P3.

T/ divide il numero triplo senza segno contenuto in P2 P3 P4 con il numero semplice senza segno contenuto in P1; il numero doppio senza segno cosi' ottenuto viene memorizzato in P1 P2. Infine viene spostato il contenuto dello Stack di due posizioni verso sinistra.

OPERAZIONI RELAZIONALI

Nel Forth gli operatori relazionali agiscono sul contenuto delle posizioni P1 e P2 dello Stack, memorizzando in P1 il risultato e spostando di una posizione verso sinistra il contenuto dello Stack. La condizione di verita' e' espressa dal valore 1, mentre quella di falso dal valore 0.

= legge i contenuti di P1 e P2 memorizzando in P1 il valore 1 se sono uguali, 0 se sono diversi.

> legge il contenuto di P1 e P2 memorizzando in P1 il valore 1 se P2>P1 mentre il valore 0 se P2<=P1.

< legge il contenuto di P1 e P2 memorizzando in P1 il valore 1 se P2<P1 mentre il valore 0 se P2>=P1.

0= legge il contenuto di P1 e vi memorizza i valori 1 o 0 a seconda che il valore originariamente contenuto fosse uguale o diverso da zero.

0< legge il contenuto di P1 e vi memorizza i valori 1 o 0 a seconda che il valore originariamente contenuto fosse minore o maggiore uguale a zero.

MIN memorizza in P1 il minore dei due valori contenuti in P1 e P2, spostando poi il contenuto dello Stack di una posizione verso sinistra.

MAX memorizza in P1 il maggiore dei due valori contenuti in P1 e P2 spostando poi il contenuto dello Stack di una posizione verso sinistra.

OPERAZIONI LOGICHE

Nel Forth gli operatori logici agiscono sulle due ultime posizioni dello Stack intervenendo su ogni singolo bit. Il risultato dell'operazione (0 oppure 1) viene memorizzato nella posizione P1.
Gli operatori logici del Forth sono:

AND memorizza in P1 il risultato dell'operazione logica AND eseguita tra i singoli bits di P1 ed i corrispondenti bits di P2.
Nella seguente tabella di verita' della funzione AND sono rappresentate le diverse coppie di valori possibili (A e B) ed il rispettivo risultato logico (C). Se ad esempio eseguiamo l'AND sui numeri 135 e 157 che corrispondono ai numeri binari:

descrizione	decimale	binario
INPUT A	135	10000111
INPUT B	157	10011101

otteniamo:

OUTPUT C	133	10000101

Tabella di verita' della funzione AND:

INPUT A	INPUT B	OUTPUT C
0	0	0
0	1	0
1	0	0
1	1	1

OR memorizza in P1 il risultato dell'operazione logica OR eseguita sulle coppie di bits di P1 e P2. La tabella di verita' della funzione OR e' la seguente:

INPUT A	INPUT B	OUTPUT C
0	0	0
0	1	1
1	0	1
1	1	1

Se ad esempio eseguiamo l'operazione OR con i due numeri 135 e 157 otteniamo:

descrizione	decimale	binario
INPUT A	135	10000111
INPUT B	157	10011101
OUTPUT C	159	10011111

XOR (EXCLUSIVE-OR) memorizza in P1 il risultato dell'operazione logica XOR eseguita sulle coppie di bits di P1 e P2. La tabella di verita' della funzione XOR e' la seguente:

INPUT A	INPUT B	OUTPUT C
0	0	0
0	1	1
1	0	1
1	1	0

Eseguendo ad esempio l'operazione XOR con i due numeri 135 e 157 otteniamo:

descrizione	decimale	binario
INPUT A	135	10000111
INPUT B	157	10011101
OUTPUT C	26	00011010

Per ragioni di chiarezza, negli esempi precedenti, sono stati impiegati numeri binari di otto bits, mentre l'interprete agisce sui bits delle locazioni P1 e P2 dello Stack, lavorando percio' su numeri di sedici bits.

Mancando nel vocabolario Forth l'utile operatore logico NOT, non implementato in questo interprete, abbiamo ritenuto utile fornire la sequenza di operazioni necessaria per ottenerla su un numero di 8 bits.
La formula impiegata e' la seguente:

N = 255 - A

in cui si ha:

A numero su cui si agisce

N risultato dell'operazione NOT eseguita su A.

La tabella di verita' della funzione e' la seguente:

NOT

INPUT A	OUTPUT B
0	1
1	0

L'operazione NOT puo' essere aggiunta al vocabolario Forth digitando:

```
: NOT 255 SWAP - ;
```

La funzione cosi' definita eseguira l'operazione logica NOT sul contenuto di P1, memorizzando il risultato sempre in P1 lasciando cosi' invariata la posizione degli altri numeri nello Stack.

Capitolo 8

MANIPOLAZIONI DELLA MEMORIA

Le istruzioni che nel Forth permettono di operare sul contenuto della memoria, sono di due tipi: il primo agisce sui singoli bytes, mentre il secondo modifica interi blocchi di memoria. Parliamo ora del primo gruppo di istruzioni.

C@ legge il valore del byte il cui indirizzo e' contenuto nella posizione P1 dello Stack.
Ad esempio, digitando:

32768 C@ . <RETURN>

Verra' visualizzato il valore 28 contenuto in nell'indirizzo 32768.

@ legge il numero semplice memorizzato all'indirizzo contenuto nella posizione P1 dello Stack ed in quello successivo.
Digitando ad esempio:

32768 @ . <RETURN>

Verra' visualizzato il numero 8732 semplice memorizzato alle locazioni 32768 e 32769.

2@ legge il numero doppio memorizzato all'indirizzo contenuto nella posizione P1 dello Stack e nei tre successivi.
Digitando ad esempio:

32768 2@ D. <RETURN>

Verra' visualizzato il numero doppio 572.282.442 memorizzato alle locazioni 32768 32769 32770 e 32771.

C! memorizza all'indirizzo contenuto in P1 il valore presente in P2.
Volendo ad esempio memorizzare il numero 154 alla locazione 5120 si operera' nel modo seguente:

154 5120 C! <RETURN>

L'istruzione C! agisce su un singolo byte, dunque il numero da memorizzare dovra' essere compreso tra 0 e 255.

! e' analoga all'istruzione C! con la sola differenza che opera su numeri semplici, cioe' su due bytes.
Vediamone ora un'applicazione:

15760 5121 ! <RETURN>

e' stato cosi' memorizzato il numero 15760 alle locazioni 5121 e 5122.

2! memorizza all'indirizzo contenuto in P1 il numero doppio contenuto in P2 P3. Ad esempio, volendo memorizzare alle locazioni 5123 5124 5125 e 5126 il numero doppio 651961, si digitera':

651961. 5123 2! <RETURN>

MANIPOLAZIONI DI BLOCCHI DI MEMORIA

Volendo intervenire su un maggior numero di bytes, anziche' su 1 2 o 4 come visto sopra, si impiegano le seguenti istruzioni.

CMOVE grazie a questa istruzione e' possibile spostare il contenuto di un blocco di memoria. Viene impiegata secondo questo schema:

```
32768  5120  1024  CMOVE  <RETURN>

  I      I      I
  I      I      I--- numero di bytes da spostare
  I      I
  I      I
  I      I---------- nuova posizione
  I
  I
  I----------------- posizione originaria
```

La posizione originaria del blocco da spostare dovra' dunque essere contenuta in P3, la nuova posizione in P2 e la lunghezza del blocco in P1.

FILL questa istruzione permette di caricare un particolare valore in tutti i bytes compresi in un blocco di memoria.

```
5120   1024  237   FILL  <RETURN>

  I       I     I
  I       I     I--- valore da caricare
  I       I
  I       I
  I       I--------- lunghezza del blocco
  I
  I
  I---------------- posizione iniziale
```

L'indirizzo del primo byte del blocco da riempire dovra' essere contenuto in P3, la lunghezza del blocco in P2 ed il valore da immettere in P1.

ERASE cancella un blocco di memoria la cui lunghezza e' specificata in P1 e il cui indirizzo del primo byte in P2.

```
5120  1024  ERASE  <RETURN>


  I      I
  I      I-- lunghezza del blocco
  I
  I
  I-------- posizione iniziale
```

Dopo aver eseguito l'istruzione ERASE, ogni byte del blocco di memoria conterra' il valore 0.

BLANKS e' analogo all'istruzione ERASE con la sola differenza che carica il valore 32 anziche' 0.

DUMP permette di visualizzare in codice esadecimale il contenuto di un blocco di memoria con la seguente procedura:

```
32768  32800  DUMP  <RETURN>


  I       I
  I       I--- indirizzo finale
  I
  I
  I---------- indirizzo iniziale
```

In P2 deve essere contenuto l'indirizzo del primo byte del blocco da visualizzare e in P1 l'indirizzo dell'ultimo byte del blocco.
Sul video vengono rappresentati gli indirizzi in codice esadecimale, il contenuto, ancora in codice esadecimale, dei quattro bytes successivi e la loro rappresentazione in reverse (codice ASCII).

TYPE permette di visualizzare (secondo il codice ASCII) i caratteri corrispondenti ai valori contenuti nei bytes di un blocco di memoria.

```
5120  100  TYPE  <RETURN>

  I     I
  I     I-- lunghezza del blocco
  I
  I
  I-------- indirizzo iniziale
```

In P2 deve essere contenuto l'indirizzo del primo byte del blocco da visualizzare e in P1 la lunghezza del blocco.
L'istruzione TYPE visualizza i caratteri corrispondenti al contenuto dei bytes impiegando il codice ASCII: nel caso incontri i valori corrispondenti, non ad un carattere, ma ad un comando, questo verra' eseguito.

Capitolo 9

LE VARIABILI E LE COSTANTI

L'interprete Forth, oltre a memorizzare dei numeri nello Stack, permette l'impiego di variabili e costanti numeriche semplici e doppie.

VARIABILI

Per assegnare un nome ad una variabile semplice, si utilizza l'apposita istruzione VARIABLE. Ad esempio, per definire la variabile di nome A, assegnandole il valore iniziale 2, si seguira' la seguente procedura:

```
2 VARIABLE A <RETURN>
```

Come si vede, si e' digitato per primo il valore da attribuire alla variabile, seguito poi dalla parola-chiave ed infine dal nome. L'interprete Forth ha cosi' assegnato il nome A ad una locazione di memoria nella quale e' contenuto il valore della variabile stessa. L'indirizzo di memoria in cui e' contenuto il valore della variabile puo' essere caricato nello Stack digitando semplicemente il nome della variabile, nel nostro caso A. In P1 verra' cosi' a trovarsi il numero della locazione di memoria in cui l'interprete ha scritto il valore 2. Per leggere o modificare il contenuto di questa particolare locazione di memoria, si impiegano le due istruzioni @ e !. Ad esempio, digitando:

```
A @ . <RETURN>
```

apparira' sul video il valore 2. Se invece volessimo modificare la variabile A, assegnandole ad esempio il valore 6, digiteremo:

```
6 A ! <RETURN>
```

Quanto detto per le variabili semplici, vale anche per quelle doppie, che agiscono su numeri doppi ovvero su quattro bytes. La sola differenza consiste nel premettere alle istruzioni gia' viste il numero 2.
Definiamo dunque la variabile doppia B:

```
37. 2VARIABLE B <RETURN>
```

Digitando ora:

```
B 2@ D. <RETURN>
```

verra' visualizzato il numero 37, valore della variabile B. L'istruzione 2! permettera' invece di modificare il valore di B:

```
56 B 2! <RETURN>
```

Si e' cosi' modificato il valore di B da 37 a 56.

COSTANTI

La procedura per definire una costante e' simile a quella gia' vista per le variabili. Esempio:

```
61 CONSTANT C <RETURN>
```

Si e' cosi' assegnato alla costante C il valore 61. Per richiamare il valore di C sara' sufficiente digitare il nome della costante seguito da <RETURN>:

```
C . <RETURN>
```

e sul video apparira' il numero 61, valore attribuito a C. Le costanti doppie vengono definite e richiamate in modo analogo. Esempio:

```
351. 2CONSTANT E <RETURN>
```

e, per richiamare il valore di E:

E D. <RETURN>

che visualizzera' il numero 351 assegnato alla costante E.

Capitolo 10

I CICLI

In questo capitolo presenteremo una delle parti piu' importanti del linguaggio: parleremo dei cicli, cioe' di quelle parole che ci permettono di ripetere piu' volte un gruppo di istruzioni, oppure di eseguirle solo se si verificano particolari condizioni. Cio' permette di manipolare la sequenza in cui vengono eseguite le istruzioni stesse.

DO ... LOOP

Il ciclo DO LOOP, analogo al FOR NEXT del Basic, permette di ripetere l'esecuzione di un gruppo di istruzioni per un determinato numero di volte. Va impiegato nel seguente modo:

```
n2 n1 DO a1 LOOP
```

dove:

- n1 : indica il valore iniziale della variabile del ciclo;
- n2 : indica il valore finale della variabile del ciclo;
- a1 : rappresenta il gruppo di istruzioni da eseguire un numero di volte uguale a n2 - n1.

La variabile del ciclo impiegata dall'interprete Forth e' identificata dal nome I e viene incrementata di 1 ogni volta che viene ripetuto il ciclo, partendo dal valore iniziale n1. L'esecuzione del ciclo termina quando la variabile I e' maggiore o uguale ad n2. Il valore di I puo' essere caricato nello Stack dando semplicemente l'istruzione I
Il diagramma di flusso del ciclo DO LOOP e' rappresentato in figura 22.

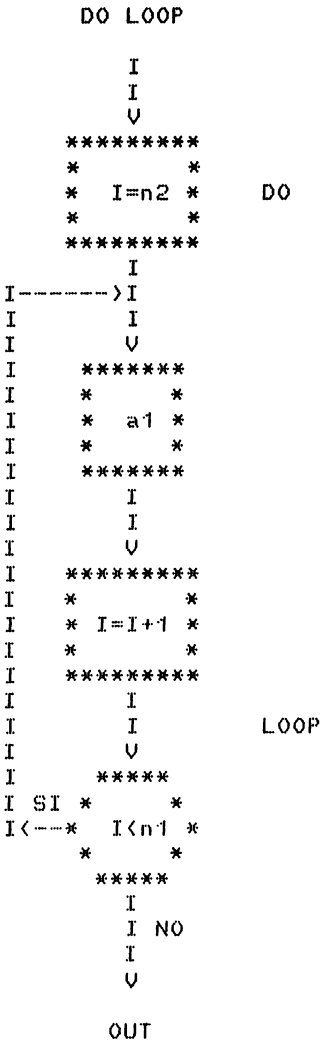


Fig. 22

Diamo ora un esempio sull'uso del ciclo DO LOOP:

```
: CICL01 CR DO I . CR LOOP ; <RETURN>
```

L'istruzione CICL01 cosi' definita esegue:

CR : posiziona il cursore all'inizio della riga successiva.

DO : apre il ciclo DO LOOP prelevando i limiti inferiore n1 e superiore n2 dalle posizioni P1 e P2 dello Stack.

I : carica nello Stack il valore attuale della variabile del ciclo.

. : visualizza sullo schermo il contenuto della posizione P1 dello Stack, ovvero il valore attuale di I.

LOOP : incrementa di 1 il valore della variabile di ciclo I e ripete l'esecuzione del ciclo fino a che I non e' maggiore o uguale al limite superiore n2. Una volta terminato il ciclo esegue le eventuali istruzioni presenti dopo la parola LOOP.

Introducendo nel calcolatore l'istruzione CICL01 nel seguente modo:

```
5 1 CICL01 <RETURN>
```

otterremo:

```
1
2
3
4
OK
```

Dall'esempio notiamo come il limite superiore n2 sia escluso dall'esecuzione.
E' pure possibile inserire un ciclo DO LOOP all'interno di altri cicli. Ad esempio con l'istruzione CICL02 che definiamo ora, la parola CICL01 viene eseguita un determinato numero di volte.

```
: CICL02 CR DO SPACE I . CR 5 1 CICL01 LOOP ;
  <RETURN>
```

L'istruzione CICL02 cosi' introdotta esegue:

```
DO      : apre il ciclo;
SPACE   : inserisce uno spazio di un carattere
CICL01  :  manda  in  esecuzione  la  parola  CICL01
          precedentemente definita;
LOOP    : chiude il ciclo.
```

Introducendo nel calcolatore l'istruzione CICL02 nel seguente modo:

```
        4 1 CICL02 <RETURN>
```

otterremo:

```
         1
        1
        2
        3
        4
         2
        1
        2
        3
        4
         3
        1
        2
        3
        4
```

E' stato cioe' ripetuto il ciclo 1 indicando a destra il valore di I del ciclo 2.

DO ... +LOOP

Il ciclo DO +LOOP, analogo al DO LOOP, permette di ripetere l'esecuzione di un gruppo di istruzioni per un determinato numero di volte, specificando pero' il valore dell'incremento della variabile di ciclo I. Va impiegato nel seguente modo:

```
n2 n1 DO a1 incr +LOOP
```

dove:

n1	: indica il valore iniziale della variabile del ciclo;
n2	: indica il valore finale della variabile del ciclo;
a1	: rappresenta il gruppo di istruzioni da eseguire un numero di volte uguale a n2 - n1;
incr	: rappresenta l'incremento da attribuire alla variabile I del ciclo. Il valore dell'incremento puo' essere anche negativo.

Il diagramma di flusso di questo ciclo e' analogo a quello del ciclo DO LOOP rappresentato in figura 22.

Diamo ora un esempio sull'impiego del ciclo DO +LOOP:

```
: CICLO3 CR DO I . CR -2 +LOOP ; <RETURN>
```

L'istruzione CICLO3 cosi' definita esegue:

DO	: apre il ciclo;
-2	: e' l'incremento assegnato alla variabile di ciclo I;
+LOOP	: chiude il ciclo.

Utilizzando la parola CICLO3 nel modo seguente:

```
-10 10 CICLO3 <RETURN>
```

otterremo:

```
10
8
6
4
2
0
-2
-4
-6
-8
OK
```

E' da notare che, se l'incremento di I e' negativo, il ciclo termina quando la variabile I ha assunto un valore minore o uguale al limite n2.

IF ... THEN

Il ciclo IF THEN, equivalente all'analogo in Basic, esegue un gruppo di istruzioni solo se si verifica una particolare condizione.
Viene impiegato nel seguente modo:

```
flag IF a1 THEN a2
```

dove:

flag : relazione logica;
a1 : istruzioni eseguite solo se la relazione logica e' vera;
a2 : eventuali istruzioni che vengono eseguite in ogni caso dopo l'IF THEN.

Il diagramma di flusso del ciclo e' rappresentato in figura 23.

Diamo ora un esempio sull'uso dell'istruzione IF THEN:

```
: CICLO4 CR IF ." CONDIZIONE VERA" THEN CR ." FINE"
  ; <RETURN>
```

dove:

IF : legge il contenuto della posizione P1 dello Stack: se e' uguale a 1 (condizione vera), esegue a1; se il contenuto di P1 e' invece 0 (condizione falsa), a1 non viene eseguito. In entrambi i casi (condizione vera o falsa) viene eseguito a2;
THEN : separa il gruppo di istruzioni a1 dalle istruzioni del gruppo a2.

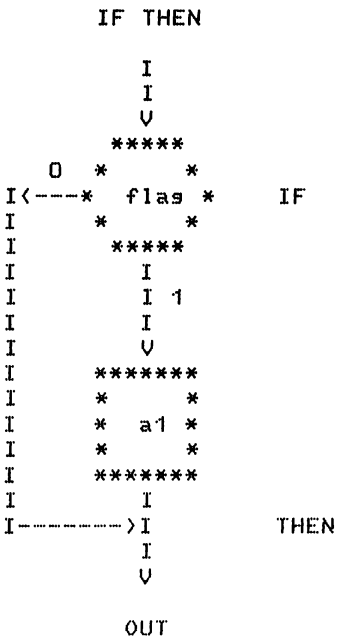


Fig. 23

Utilizzando la parola CICL04 nel modo seguente:

3 3 = CICL04 <RETURN>

otterremo:

CONDIZIONE VERA
FINEOK

Infatti, essendo vero che 3=3, il risultato dell'operazione di confronto e' 1; viene quindi eseguito il gruppo di istruzioni a1, ovvero visualizzata la scritta "condizione vera". Quando la condizione logica e' invece falsa, come nell'esempio seguente, si avra':

```
3 2 = CICLO4 <RETURN>
```

otterremo:

```
FINEOK
```

Essendo cioe' la condizione logica falsa, non e' stato eseguito il gruppo di istruzioni a1, ma solo il gruppo a2. In a1 e a2 possono essere contenute istruzioni di ogni tipo, anche cicli IF THEN, purche' completi.

IF ... ELSE ... THEN

Differisce dal precedente ciclo IF THEN perche' permette di eseguire un gruppo di istruzioni solo se la condizione logica precedente l'IF e' falsa. Viene impiegato nel seguente modo:

```
flag IF a1 ELSE a2 THEN a3
```

dove:

flag : condizione logica;
a1 : istruzioni eseguite solo se la condizione logica e' vera;
a2 : istruzioni eseguite solo se la condizione logica e' falsa;
a3 : eventuali istruzioni che vengono eseguite in ogni caso dopo l'IF THEN.

Il diagramma di flusso del ciclo e' rappresentato in figura 24.

IF ELSE THEN

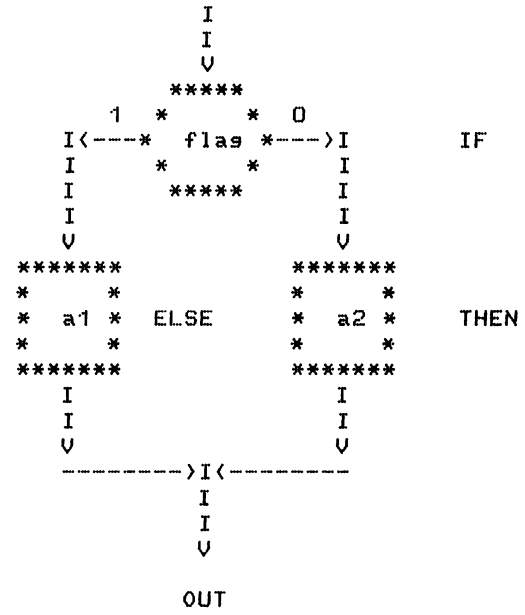


Fig. 24

Diamo ora un esempio sull'uso dell'istruzione IF ELSE THEN:

```
: CICLO5 CR IF ." CONDIZIONE VERA" ELSE ."
  CONDIZIONE FALSA" THEN CR ." FINE" ; (RETURN)
```

dove:

IF : legge il contenuto della posizione P1 dello Stack: se e' uguale a 1 (condizione vera), esegue a1; se il contenuto di P1 e' invece 0 (condizione falsa), viene eseguito a2. In entrambi i casi (condizione vera o falsa) viene eseguito a3.

```
        ELSE    : separa  il gruppo  di istruzioni a1 dalle
                  istruzioni del gruppo a2.
        THEN    : separa  il gruppo  di istruzioni a2 dalle
                  istruzioni del gruppo a3.
```

Utilizzando la parola CICLO5 nel modo seguente:

```
        3 3 = CICLO5 <RETURN>
```

otterremo:

```
        CONDIZIONE VERA
        FINEOK
```

Utilizzandola invece nel modo seguente:

```
        3 2 = CICLO5 <RETURN>
```

otterremo:

```
        CONDIZIONE FALSA
        FINEOK
```

BEGIN ... AGAIN

Il ciclo BEGIN AGAIN ripete continuamente l'esecuzione di un gruppo di istruzioni senza permettere l'uscita dal ciclo stesso. Il ciclo e' cosi' strutturato:

```
        BEGIN a1 AGAIN
```

dove:

```
        BEGIN : apre il ciclo;
        a1    : gruppo  di istruzioni  ripetue durante.  il
                ciclo;
        AGAIN : chiude il ciclo.
```

Nella fisura 25 e' rappresentato il diagramma di flusso del ciclo BEGIN AGAIN.

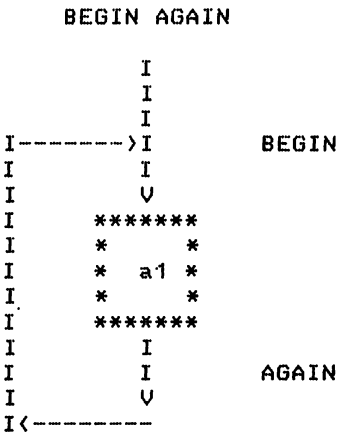


Fig. 25

Forniamo di seguito un esempio sull'uso del ciclo BEGIN AGAIN:

```
: CICL06 CR ." INIZIO" BEGIN ." CONTINUA" CR AGAIN
  ; <RETURN>
```

Dopo averne introdotto la definizione, digitando la parola CICL06, verra' visualizzato sul video:

```
CICL06 <RETURN>
INIZIO
CONTINUA
CONTINUA
CONTINUA
CONTINUA
...
...
```

Cioe' si avra' un'esecuzione continua delle istruzioni a1, nel caso specifico la visualizzazione della scritta CONTINUA.

BEGIN ... UNTIL

Ripete continuamente l'esecuzione di un gruppo di istruzioni fino a quando non si verifica una particolare condizione. Struttura del ciclo:

```
BEGIN a1 flag UNTIL a2
```

dove:

BEGIN : apre il ciclo;
a1 : gruppo di istruzioni eseguite nel ciclo;
flag : operazione logica che da 0 o 1 a seconda che il risultato sia vero o falso;
UNTIL : legge il contenuto di P1: se e' 0 ripete il ciclo; se invece e' 1 esce dal ciclo stesso;
a2 : istruzioni che vengono eseguite dopo l'uscita dal ciclo.

Il diagramma di flusso del ciclo viene rappresentato nella figura 26.

Forniamo un esempio sull'uso del ciclo BEGIN UNTIL: introdotta la parola CICLO7, iniziera' un ciclo che sara' interrotto solo premendo il tasto <E>.

```
: CICLO7 BEGIN KEY 69 = UNTIL CR ." FINE" ;
  <RETURN>
```

dove:

BEGIN : apre il ciclo;
KEY : attende che venga premuto un tasto e carica nello Stack il valore corrispondente secondo il codice ASCII;
69 : e' il valore corrispondente al tasto <E> secondo il codice ASCII;

= confronta il valore caricato nello Stack dall'istruzione KEY con il numero 69, caricando in P1 il valore 1 oppure 0 a seconda che la condizione sia vera o falsa.

UNTIL : legge il contenuto di P1: se e' 0 ripete il ciclo; se e' 1 esce dal ciclo.

Digitando dunque la parola CICLO7, otterremo:

```
CICLO7 <RETURN>
```

e, premendo <E>

```
FINEOK
```

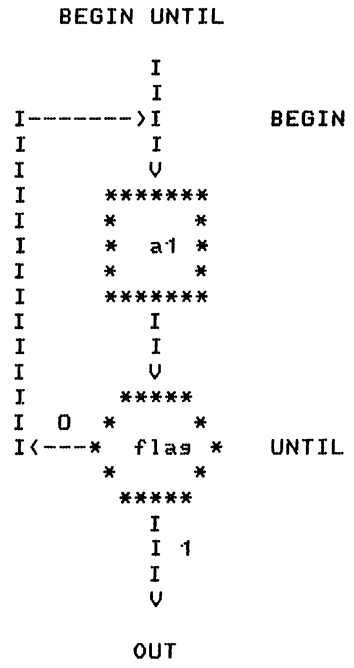


Fig. 26

BEGIN ... WHILE ... REPEAT

Il ciclo BEGIN WHILE REPEAT e' simile al BEGIN UNTIL con la differenza che non esamina il flag alla fine del ciclo, ma prima della parola WHILE. La sua configurazione e' la seguente:

```
BEGIN a1 flag WHILE a2 REPEAT a3
```

dove:

```
BEGIN : apre il ciclo;
a1    : gruppo di istruzioni eseguite nella prima
        parte del ciclo;
flag  : condizione logica;
WHILE : esamina il flag uscendo dal ciclo se e'
        falso;
a2    : secondo gruppo di istruzioni del ciclo;
REPEAT : chiude il ciclo;
a3    : gruppo di istruzioni eseguite dopo l'uscita
        dal ciclo.
```

La figura 27 rappresenta il diagramma di flusso del ciclo.

Diamo di seguito un esempio di impiego del ciclo sopra visto:

```
: C8 ." COME VA ?" BEGIN KEY 66 = WHILE CR ." VA
  BENE" REPEAT CR ." VA MALE" ; <RETURN>
```

dove:

```
BEGIN : apre il ciclo;
WHILE : legge il contenuto di P1: se e' 1
        (condizione falsa), continua il ciclo; se e'
        0 (condizione vera), esce dal ciclo.
REPEAT: chiude il ciclo;
```

Utilizzando la parola C8 otterremo, premendo il tasto <B>, la scritta VA BENE; premendo un qualsiasi altro tasto apparira' invece la scritta VA MALE. Infatti:

BEGIN WHILE REPEAT

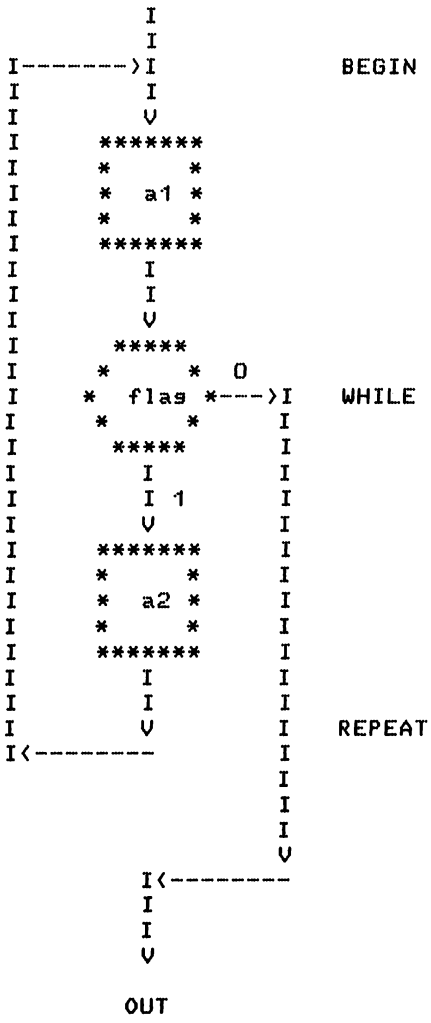


Fig. 27

CB <RETURN>

si avra'

COME VA ?

e, premendo ogni volta <B>

VA BENE
VA BENE
VA BENE
...
...

premendo invece un altro tasto, apparira'

VA MALEOK

Capitolo 11

METODO DI PROGRAMMAZIONE DEL FORTH

Dopo aver presentato nei precedenti capitoli le basi per poter programmare in Forth, forniamo ora alcuni significativi programmi in grado di chiarire meglio le funzioni e l'impiego dei concetti appresi. Abbiamo realizato dei programmi sulla gestione di stringhe, uso per cui il Forth si presta particolarmente, viste le sue caratteristiche di velocita' e potenza.

MEMORIZZAZIONE DI STRINGHE

Il primo programma, cui e' stato attribuito il nome IN$, permette di immettere delle stringhe alfanumeriche in una particolare zona di memoria scelta dal programmatore. Seguira', dopo il listato del programma, la spiegazione del suo funzionamento e della sua struttura. Esso va impiegato nel seguente modo: verra' prima immesso l'indirizzo della locazione di memoria in cui iniziera' la memorizzazione della stringa; seguira' poi la parola IN$ e <RETURN>. Il calcolatore sara' cosi' pronto a ricevere la stringa direttamente da tastiera. Il <RETURN> finale segnera' la fine della stringa stessa.

LISTATO DEL PROGRAMMA IN$

```
100 VARIABLE LM <RETURN>

0 VARIABLE T <RETURN>

0 VARIABLE P <RETURN>

: L? P @ T @ < IF ELSE DROP 13 DUP P @ C! 1 P +! THEN
  ;<RETURN>

: IN$ CR DUP P ! LM @ + T ! BEGIN KEY DUP EMIT DUP P @ ! 1
        P    +!    L?    13    =    UNTIL    ;    <RETURN>
```

DIAGRAMMA DI FLUSSO DEL PROGRAMMA IN$

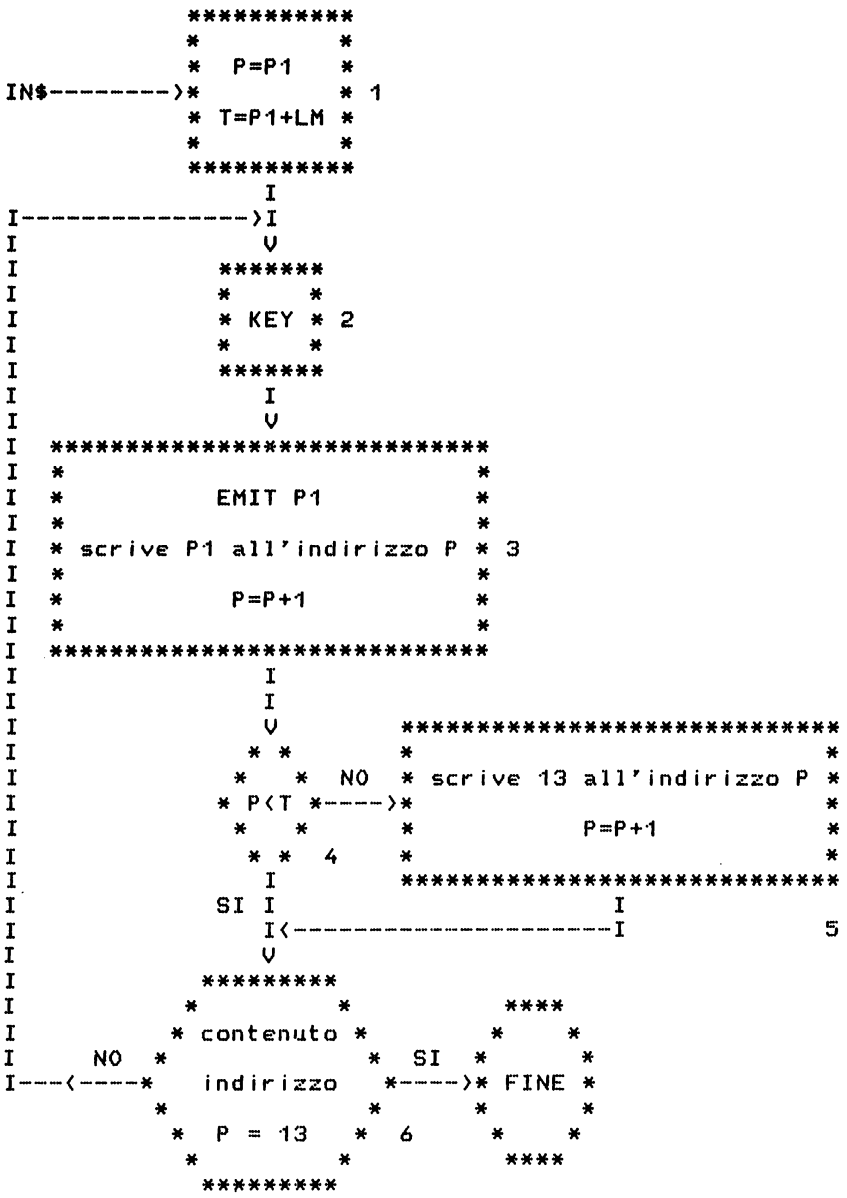


Fig. 28

Dal diagramma di flusso di figura 28 vediamo la struttura del programma IN$.
Nel blocco numero 1, il valore letto nella posizione P1 dello Stack viene assegnato alla variabile P; alla variabile T viene invece assegnato il valore di P sommato a quello di LM, lunghezza massima della stringa. Il valore di P rappresenta cosi' l'indirizzo a cui viene iniziata la memorizzazione della stringa, mentre T quello entro il quale la stringa termina. Il numero massimo dei caratteri che possono essere contenuti nella stringa, definito da LM, puo' essere modificato a piacere compatibilmente alla disponibilita' di memoria.
Con il blocco 2 inizia il ciclo di input dei caratteri da tastiera: l'istruzione KEY provoca un blocco del sistema fino alla digitazione di un carattere, il cui corrispondente valore in codice ASCII verra' memorizzato nella posizione P1 dello Stack.
Il blocco 3 ha il compito di visualizzare sullo schermo il carattere digitato e di memorizzarne il valore corrispondente all'indirizzo P. Infine incrementa il valore P di uno.
Nel blocco 4 viene verificato se P e' minore di T, cioe' se e' stato raggiunto il numero massimo di caratteri della stringa. Nel caso P non sia minore di T, si passa al blocco 5, dove viene simulata la digitazione del tasto <RETURN>, caricando il valore corrispondente (13) all'indirizzo P e incrementando poi P di uno.
Nel blocco 6 viene letto il contenuto dell'indirizzo P: se e' diverso da 13 si ricomincia il ciclo di input dal blocco 2. Se e' uguale a 13 si esce dal ciclo.

Per rendere piu' semplice il programma, esso e' stato spezzato in due parti: la prima, costituita dai blocchi 4 e 5, e' contenuta nella parola L?; la seconda, costituita dagli altri blocchi, si trova nella parola IN$.
Prima di immettere queste due parole sono state definite le variabili utilizzate, assegnando loro i valori iniziali.
L'istruzione L? e' costituita da un ciclo IF ELSE THEN il cui flag e' dato dal confronto tra i valori di P e di T; il blocco 6 corrisponde alle istruzioni comprese tra ELSE e THEN.
La parola IN$, dopo aver eseguito le operazioni del blocco 1, inizia un ciclo BEGIN UNTIL che esegue i blocchi 2 e 3, la parola L? e il blocco 6 (che fornisce il flag per la prosecuzione del ciclo).

VISUALIZZAZIONE DI STRINGHE

Il secondo programma, che prende il nome di OUT$, permette di visualizzare delle stringhe alfanumeriche memorizzate in una particolare zona di memoria. Il programma OUT$, da usarsi in abbinamento al precedente IN$, va impiegato nel seguente modo: verra' immesso l'indirizzo della prima locazione di memoria in cui e' memorizzata la stringa; seguira' poi la parola OUT$ e <RETURN>. Il calcolatore visualizzera' la stringa richiesta.
Diamo la spiegazione del funzionamento e della struttura del programma dopo il seguente listato.

LISTATO DEL PROGRAMMA OUT$

```
: F? P @ T @ < IF ELSE DROP 13 1 P +! THEN ; <RETURN>

: OUT$ CR DUP P ! LM @ + T ! BEGIN P @ C@ DUP  EMIT 1 P +!
  F? 13 = UNTIL ; <RETURN>
```

Dal diagramma di flusso di figura 29 vediamo la struttura del programma OUT$.
Nel blocco numero 1, il valore letto nella posizione P1 dello Stack viene assegnato alla variabile P; alla variabile T viene invece assegnato il valore di P sommato a quello di LM, lunghezza massima della stringa. Il valore di P rappresenta cosi' l'indirizzo a cui viene iniziata la lettura della stringa, mentre T quello entro il quale la stringa termina.
Con il blocco 2 inizia il ciclo di visualizzazione della stringa: l'istruzione EMIT visualizza il carattere corrispondente al numero letto all'indirizzo P, incrementando poi P di uno.
Nel blocco 3 viene verificato se P e' minore di T, cioe' se e' stata superata la lunghezza massima della stringa. Nel caso P non sia minore di T, si passa al blocco 4, dove viene caricato il numero 13 (corrispondente, in codice ASCII, al tasto <RETURN>), e incrementato P di uno.
Nel blocco 5 viene letto il contenuto della posizione P1 dello Stack: se il numero letto e' diverso da 13, si ricomincia il ciclo di lettura dal blocco 2. Se invece e' uguale a 13 si esce dal ciclo.

DIAGRAMMA DI FLUSSO DEL PROGRAMMA OUT$

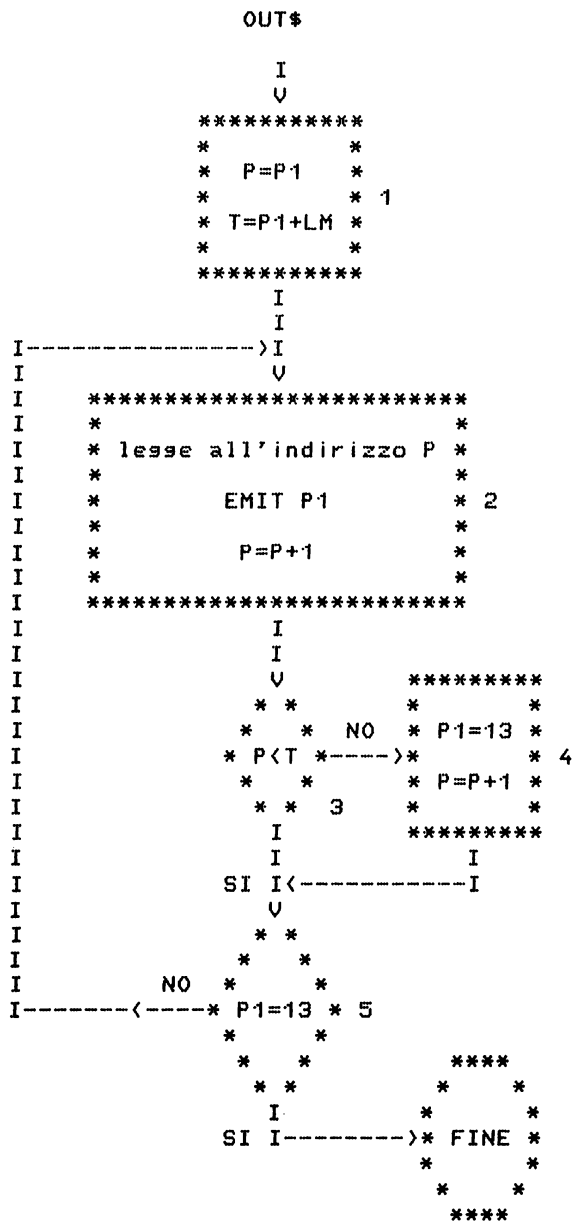


Fig. 29

Il programma e' composto dalle due parole F? e OUT$: la prima e' costituita dai blocchi 3 e 4, mentre la seconda dai restanti blocchi.
L'istruzione F? e' costituita da un ciclo IF ELSE THEN il cui flag e' dato dal confronto tra i valori di P e di T; il blocco 5 corrisponde alle istruzioni comprese tra ELSE e THEN.
La parola OUT$, dopo aver eseguito le operazioni del blocco 1, inizia un ciclo BEGIN UNTIL che esegue il blocco 2, la parola F? e il blocco 5 (che fornisce il flag per la prosecuzione del ciclo).

CONFRONTO DI STRINGHE

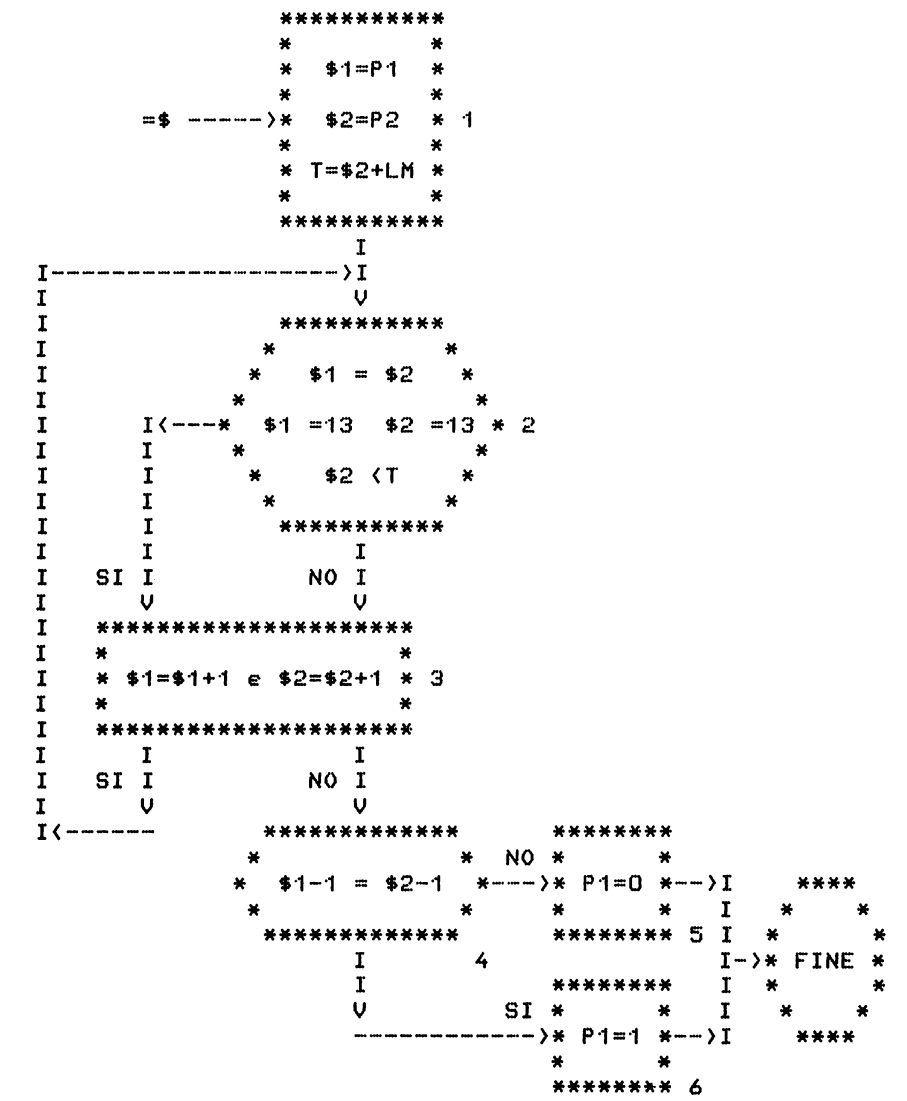
Questo programma, di nome =$, permette di confrontare due stringhe alfanumeriche e di verificare se sono uguali: va usato insieme ai due precedenti programmi IN$ e OUT$ e deve essere impiegato nel seguente modo: si introducono gli indirizzi delle due stringhe da confrontare; seguira' poi la parola =$ e <RETURN>. Verranno cosi' confrontate le due stringhe che iniziano agli indirizzi immessi. Ultimato il confronto verra' posto, nella posizione P1 dello Stack, uno 0 o un 1 a seconda che le due stringhe siano diverse o uguali.

Diamo la spiegazione del funzionamento e della struttura del programma dopo il seguente listato.

LISTATO DEL PROGRAMMA =$

```
0 VARIABLE $1 <RETURN>

0 VARIABLE $2 <RETURN>

: E? 2DUP = $2 @ T @ < AND 1 SWAP 2SWAP 13 = SWAP 13 = OR 1
  SWAP - AND - ; <RETURN>

: SUB $1 @ C@ $2 @ C@ E? 1 $1 +! 1 $2 +! ; <RETURN>

: =$ $1 ! DUP $2 ! LM @ + T ! BEGIN SUB UNTIL $1 @ 1- C@ $2
  @ 1- C@ = ; <RETURN>
```

DIAGRAMMA DI FLUSSO DEL PROGRAMMA =$



$1 = valore del byte di indirizzo $1
$2 = valore del bYte di indirizzo $2

Fig. 30

Dal diagramma di flusso di figura 30 vediamo la struttura del programma =$.
Nel blocco numero 1, alla variabile $1 viene assegnato il valore letto nella posizione P1 dello Stack; alla variabile $2 quello letto nella posizione P2. T viene posto uguale a $2 piu' LM. Le variabili $1 e $2 rappresentano gli indirizzi a cui iniziano le stringhe che devono essere confrontate, mentre T quello entro il quale la stringa $2 termina.
Con il blocco 2 inizia il ciclo di confronto delle stringhe: vengono confrontati i caratteri memorizzati agli indirizzi $1 e $2 e si verifica che $2 sia minore di T (ovvero che non sia stata superata la lunghezza massima di $2). Viene inoltre verificato, confrontando i caratteri letti con il numero 13 (valore corrispondente, in codice ASCII, al <RETURN>), se si e' giunti alla fine di una delle due stringhe.
Si passa quindi al blocco 3, dove $1 e $2 vengono incrementati di 1. Se i confronti effettuati nel blocco 2 sono tutti positivi, si ritornera' al blocco 2; in caso contrario si passera' al blocco 4.
Quest'ultimo blocco confronta tra loro i contenuti dei bytes di indirizzo $1-1 e $2-1. Se sono diversi si passa al blocco 5; se, al contrario, sono uguali al 6.
Il blocco 5 carica uno 0 nella posizione P1 dello Stack, mentre il blocco 6 carica un 1.

Il programma =$ e' costituito dalle tre parole E? SUB e =$ e contiene inoltre la definizione delle variabili $1 e $2.
La parola E? corrisponde al blocco 2. Confronta il contenuto dei bytes di indirizzo $1 e $2, verifica se $2 e' minore di T ed esegue l'operazione AND tra i due risultati. Inoltre vengono confrontati i contenuti dei bytes $1 e $2 con il valore 13 eseguendo poi l'operazione logica OR tra i due risultati. La sequenza (1 SWAP -) esegue un NOT sul risultato dell'operazione OR precedente. Viene poi effettuata un'operazione AND tra il risultato del NOT e quello del primo AND. Sul risultato viene in fine effettuata un'operazione NOT. Questa sequenza di operazioni puo' essere scritta matematicamente nel seguente modo:

NOT(($1 = $2)AND($2(T))AND(NOT(($1 =13)OR($2 =13)))

Nella parola SUB viene eseguita l'istruzione E? ed il blocco 3.
La parola =$, dopo aver svolto le operazioni del blocco 1, inizia un ciclo BEGIN UNTIL in cui viene eseguita la parola SUB.

Il flag di questo ciclo e' costituito dalla parola E?, ed e' 0 solo se i confronti effettuati nel blocco 2 sono tutti positivi, ovvero se i due caratteri di indirizzo $1 e $2 sono uguali, il valore di $2 e' minore di T ed i bytes di indirizzo $1 e $2 sono diversi da 13.
Quando il flag e' 1 si esce dal ciclo BEGIN UNTIL e si eseguono i blocchi 4 5 e 6, effettuando di nuovo il confronto tra gli ultimi due caratteri letti dalle stringhe. Se i due caratteri sono uguali, significa che si e' usciti dal ciclo perche' e' stato ultimato il confronto tra le due stringhe e queste sono risultate uguali. Se i due caratteri confrontati sono diversi significa invece che le stringhe sono diverse.

MISURA DELLE STRINGHE

Questo programma, di nome L$, permette di contare i caratteri alfanumerici di una stringa, e va usato insieme ai precedenti programmi. Si impiega nel seguente modo: si introduce l'indirizzo della stringa da misurare; seguira' poi la parola L$ e <RETURN>. Verranno cosi' contati i caratteri della stringa che inizia all'indirizzo immesso. Ultimato il conteggio, verra' posta nella posizione P1 dello Stack la lunghezza della stringa.

Diamo ora il listato del programma. Seguiranno la spiegazione del suo funzionamento e della sua struttura.

LISTATO DEL PROGRAMMA L$

```
: L$ DUP $1 ! 1- T ! BEGIN 1 T +! T @ C@ 13 = UNTIL T @ $1
  @ - ; <RETURN>
```

Dal diagramma di flusso di figura 31 viene illustrata la struttura del programma L$.
Nel blocco numero 1, alla variabile $1 viene assegnato il valore letto nella posizione P1 dello Stack; la variabile T viene posta uguale al valore ($1-1). La variabile $1 rappresenta l'inizio della stringa, mentre T quello in cui la stringa termina.
Con il blocco 2 inizia il ciclo di misurazione della stringa incrementando di 1 la variabile T.

DIAGRAMMA DI FLUSSO DEL PROGRAMMA L$

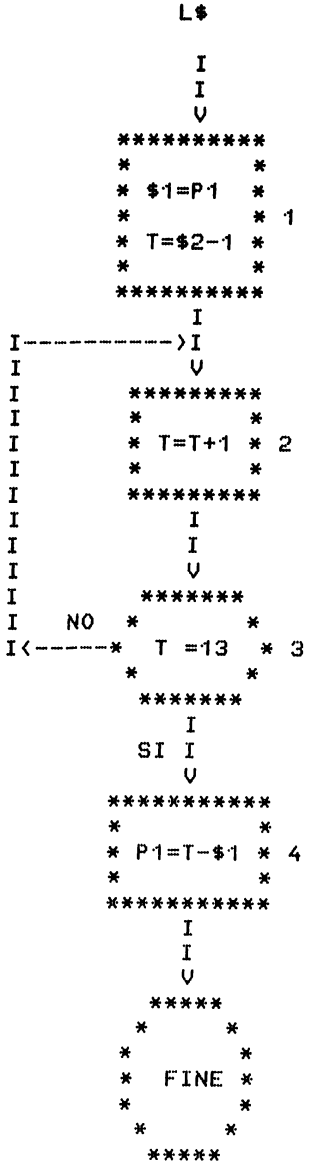


Fig. 31

Il blocco 3 confronta il contenuto del byte di indirizzo T con il valore 13. Se i valori sono diversi, si ritorna al blocco 2; se al contrario sono uguali, si passa al blocco 4.
Quest'ultimo blocco pone nella posizione P1 dello Stack il risultato dell'operazione (T-$1), corrispondente alla lunghezza della stringa.

La parola L$, dopo aver eseguito le operazioni del blocco 1, inizia un ciclo BEGIN UNTIL in cui esegue il blocco 2. Il flag di questo ciclo e' costituito dal blocco 3. Viene infine eseguita l'operazione di differenza del blocco 4, che da come risultato la lunghezza della stringa.

RICERCA DI STRINGHE

In molti casi si presenta la necessita', in programmi ad esempio di tipo gestionale o di archivio, di ricercare all'interno di una serie di dati una particolare stringa alfanumerica. In questo programma vi presentiamo una possibile soluzione del problema: nel caso particolare, sfruttando il fatto che nel Forth e' possibile definire una stringa la cui lunghezza e' limitata solo dalla quantita' di memoria disponibile, e' stata creata un'unica stringa nella quale memorizzare i dati, dando la possibilita' di ricercarli poi separatamente. Il problema e' stato ricondotto, in sostanza, alla ricerca di una stringa (dato da ricercare) all'interno di un'altra (dati gia' memorizzati).

Il programma e' stato chiamato R$, e va impiegato nel seguente modo: vanno immessi prima l'indirizzo a cui inizia la stringa contenente i dati, poi l'indirizzo della stringa da ricercare. Infine si digita l'istruzione R$ e <RETURN>. Dopo che la ricerca e' stata ultimata, nella posizione P1 dello Stack verra' posto un 1 se la stringa e' stata trovata, oppure uno 0 nel caso contrario. Se in P1 si trova un 1, la variabile T2 conterra' l'indirizzo dove inizia, all'interno della stringa-dati, una stringa uguale a quella cercata.

Diamo ora il listato del programma. Seguiranno la spiegazione del suo funzionamento e della sua struttura.

LISTATO DEL PROGRAMMA R$

```
0 VARIABLE T1 <RETURN>

0 VARIABLE T2 <RETURN>

0 VARIABLE LL <RETURN>

: S4 1 $1 +! 1 $2 +! ; <RETURN>

: S3 BEGIN $1 @ C@ DUP $2 @ C@ = SWAP 13 = 1 SWAP - AND
  WHILE S4 REPEAT ; <RETURN>

: S2 T1 @ $1 ! T2 @ $2 ! S3 $1 @ C@ 13 = ; <RETURN>

: S1 DUP T2 ! DUP L$ + LL ! T1 ! ; <RETURN>

: R$ S1 BEGIN S2 1 SWAP - T2 @ LL @ < AND WHILE 1 T2 +!
  REPEAT S2 ; <RETURN>
```

Dal diagramma di flusso di figura 32 viene illustrata la struttura del programma R$.

Per ragioni di chiarezza il diagramma di flusso e' stato diviso in due parti di cui una, che nello schema prende il nome di A (fig. 33), viene ripetuta due volte. Il blocco A agisce in questo modo: la stringa da ricercare (che abbrevieremo con SR), viene confrontata con la stringa-dati (SD), partendo dall'inizio delle stringhe. Il primo carattere di SR viene confrontato con il primo di SD. Se sono uguali, verranno confrontati i secondi due e cosi' via fino a che o viene trovato un carattere diverso o e' terminata la stringa SR. Questo confronto viene effettuato dal blocco A nel seguente modo:
come vediamo dalla figura 33, nel blocco 4 viene posto $1=T1 e $2=T2. T1 e T2 contengono gli indirizzi a cui iniziano le due stringhe.
I due blocchi successivi (numeri 5 e 6), si occupano, tramite un ciclo, del confronto tra le due stringhe: dal blocco 6, se i due caratteri di indirizzo $1 e $2 sono uguali e se in $1 non e' presente il numero 13 (corrispondente in codice ASCII al carattere <RETURN>), si passa al blocco 5, dove $1 e $2 vengono incrementati di 1.

DIAGRAMMA DI FLUSSO DEL PROGRAMMA R$

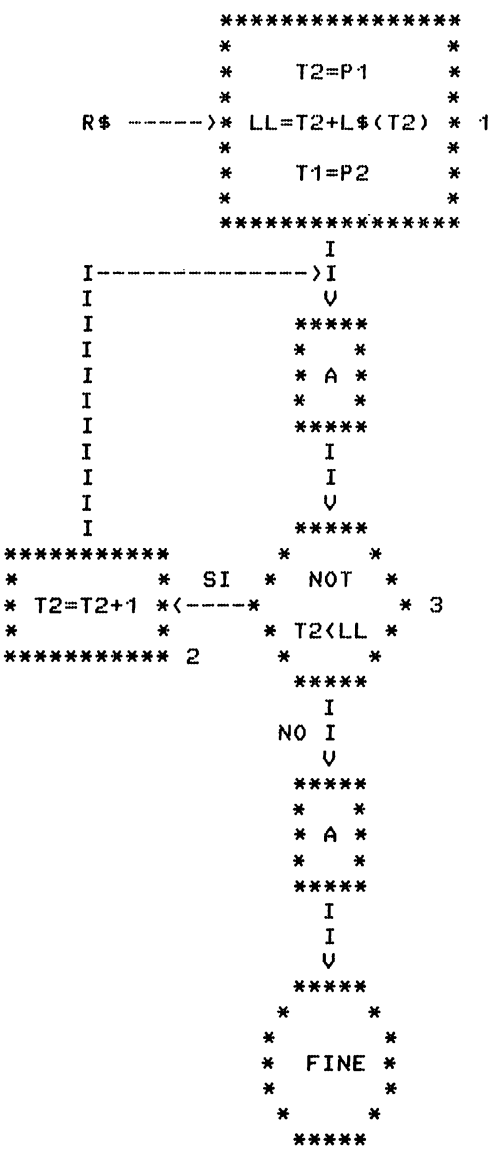


Fig. 32

DIAGRAMMA DI FLUSSO DEL BLOCCO A

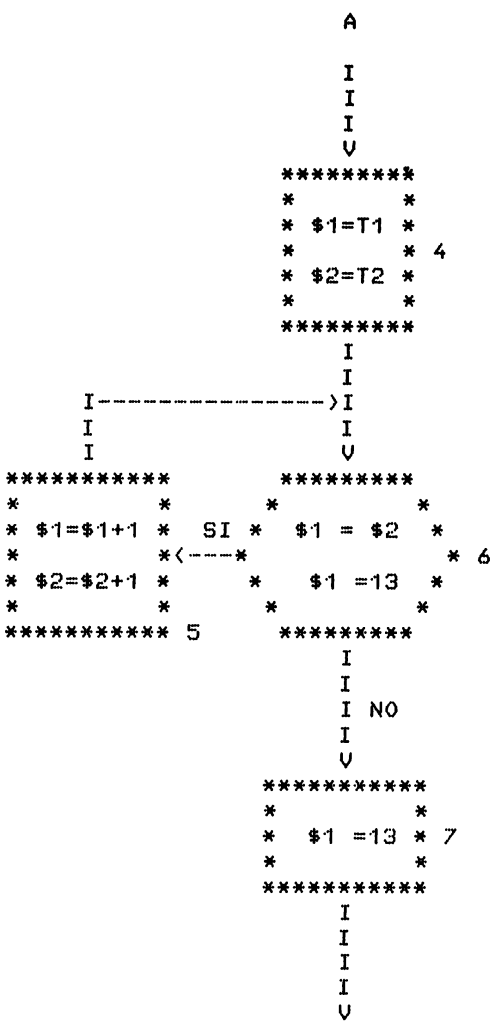


Fig. 33

Si ritorna poi di nuovo al blocco 6. Con questo ciclo vengono dunque confrontate, carattere per carattere, le due stringhe SR e SD, fino a quando o viene trovato un carattere diverso o la stringa SR e' terminata. Il flag presente nel blocco 7 permette di verificare se il confronto e' stato ultimato ovvero se le stringhe sono uguali.

Passiamo ora alla figura 32 in cui e' rappresentato il diagramma di flusso complessivo.
Nel blocco 1 vengono letti dallo Stack i valori da assegnare alle variabili T1 e T2. Nella variabile LL viene posto l'indirizzo a cui termina la stringa SD.
Con i blocchi A, 2 e 3 si esegue il ciclo di ricerca di SR in SD.
Il blocco 3 causa l'uscita dal ciclo nel caso che il flag del blocco A sia 1 oppure che la stringa SD sia terminata. In caso contrario si passa al blocco 2 dove, incrementando T2 di 1, la ricerca di SR in SD verra' spostata avanti di una posizione.
Terminato il ciclo verra' posto nello Stack un 1 o uno 0 (con l'esecuzione del ciclo A), a seconda che la stringa sia o meno stata trovata. La variabile T2 indichera' la posizione in SD dove e' localizzata SR.

Il blocco A viene eseguito dalla parola S2 che, dopo aver svolto le operazioni del blocco 4, richiama la parola S3 per poi eseguire il flag del blocco 7.
In S3 viene svolto il blocco 6 tramite un ciclo BEGIN WHILE REPEAT ed il blocco 5, richiamando la parola S4 dopo il WHILE.
La parola R$ esegue il blocco 1 richiamando S1, e svolge un ciclo BEGIN WHILE REPEAT di ricerca, in cui vengono svolte le seguenti operazioni:
1) esecuzione della parola S2 e del blocco 3 (che fornisce il flag per l'uscita dal ciclo);
2) esecuzione, dopo il WHILE, del blocco 2;
3) esecuzione, all'uscita dal ciclo di ricerca, della parola S2.

Capitolo 12

COME LAVORA IL FORTH

Vista l'importanza che presenta nel Forth l'uso e la manipolazione diretta da parte del programmatore della memoria, presentiamo in questo capitolo la sua mappa ed ulteriori istruzioni per agire su di essa.
Come vediamo dalla mappa di memoria di figura 34, possedendo il cartridge del Forth un'espansione di memoria RAM di 3K, il sistema si riconfigura spostando la mappa schermo all'indirizzo 4096 (esadecimale 1000) e la mappa colore all'indirizzo 37888 (esa. 9400). I 3K di espansione RAM si posizionano dall'indirizzo 1024 (esa. 0400) all'indirizzo 4095 (esa. OFFF), mentre gli 8K di espansione ROM dall'indirizzo 40960 (esa. A000) all'indirizzo 49151 (esa. BFFF).
L'interprete Forth memorizza la definizione delle nuove parole introdotte all'indirizzo contenuto nella variabile DP, corrispondente all'indirizzo del primo byte libero dopo il vocabolario. Al momento dell'accensione DP assume il valore 4806. Definendo una nuova parola l'interprete Forth altera opportunamente il valore di DP, che puo' essere letto con l'istruzione HERE o modificato con l'istruzione ALLOT nel seguente modo:

```
n ALLOT <RETURN>
```

dove n e' il numero semplice con segno che verra' sommato a DP. E' cosi' possibile memorizzare la definizione di una parola in una particolare zona della memoria o lasciare delle locazioni di memoria libere all'interno del vocabolario creato dal programmatore. Queste zone di memoria possono, ad esempio, essere utilizzate per memorizzare dei dati o delle stringhe che verranno salvate su nastro o su disco insieme al vocabolario stesso.

La zona di memoria normalmente utilizzata per la memorizzazione dei programmi va dall'indirizzo 4806 fino alla fine della RAM utente che, nella configurazione base, termina all'indirizzo 8191 (esa. OFFF).Nella zona di memoria compresa tra gli indirizzi 1024 (esa. 0400) e 4095 (esa. OFFF) e' generalmente possibile operare con le istruzioni viste nel capitolo 8 sulla manipolazione della memoria, senza provocare un blocco del sistema.

MAPPA DI MEMORIA

Indirizzi dec.	Indirizzi esa.	Funzione
0	0000	1K RAM area di lavoro per Sistema Operativo e Forth
1023	03FF	
1024	0400	3K RAM espansione contenuta nel cartridge
4095	0FFF	
4096	1000	0.5K RAM mappa schermo
4607	01FF	
4608	0200	3.5K RAM utente
8191	1FFF	
8192	2000	8K RAM/ROM espansione
16383	3FFF	
16384	4000	8K RAM/ROM espansione
24575	5FFF	
24576	6000	8K RAM/ROM espansione
32767	7FFF	
32768	8000	4K ROM mappa caratteri
36863	8FFF	
36864	9000	1K RAM Input/Output 0
37877	93FF	
37888	9400	0.5K RAM mappa colore
38399	95FF	
38400	9600	0.5K RAM
38911	97FF	
38912	9800	1K RAM Input/Output 2
38935	98FF	
38936	9C00	1K RAM Input/Output 3
40959	9FFF	
40960	AFFF	8K ROM interprete Forth
49151	BFFF	
49512	C000	8K ROM interprete Basic
57343	DFFF	
57344	E000	8K ROM sistema operativo
65535	FFFF	

Fig. 34

Esaminiamo ora come vengono memorizzate le definizioni delle nuove parole introdotte dal programmatore.

Nell'indirizzo DP (che corrisponde all'indirizzo nfa della nuova parola), all'introduzione di una nuova parola verra' memorizzato un numero binario indicante la lunghezza del nome assegnato alla parola stessa.

Nei bytes successivi viene memorizzato:

1) il nome della parola (tramite il codice ASCII), utilizzando un byte per ogni lettera;
2) l'indirizzo nfa relativo alla parola precedentemente definita, utilizzando due bytes (il cui indirizzo prende il nome di lfa);
3) il codice relativo all'istruzione introdotta utilizzando due bytes (il cui indirizzo si chiama cfa);
4) la definizione della parola introdotta (a partire dall'indirizzo di nome pfa), memorizzando come numeri semplici gli indirizzi nfa relativi alle parole che la definiscono;
5) un numero semplice indicante la fine della definizione della parola.

Nello schema seguente illustriamo il modo con cui e' possibile, partendo dal nome della parola, risalire agli indirizzi di definizione.

istruzione ' (fornisce l'indirizzo pfa):

' nome parola <RETURN>

istruzione NFA (fornisce l'indirizzo nfa):

indirizzo pfa NFA <RETURN>

istruzione LFA (fornisce l'indirizzo lfa):

indirizzo pfa LFA <RETURN>

istruzione CFA (fornisce l'indirizzo cfa):

indirizzo pfa CFA <RETURN>

istruzione PFA (fornisce l'indirizzo pfa):

indirizzo nfa PFA <RETURN>

istruzione ID. (fornisce il nome dell'istruzione)

indirizzo nfa ID. <RETURN>

Capitolo 13

IL SUONO

Per la gestione del suono l'interprete Forth utilizza, nel VIC 20, gli stessi registri che vengono impiegati dall'interprete Basic.
Diamo qui di seguito gli indirizzi delle locazioni di memoria da modificare per la generazione dei suoni:

indirizzi	valore	funzione
36874	X	valore di X compreso tra 128 e 255; seleziona la frequenza del primo oscillatore sonoro;
36875	X	valore di X compreso tra 128 e 255; seleziona la frequenza del secondo oscillatore sonoro;
36876	X	valore di X compreso tra 128 e 255; seleziona la frequenza del terzo oscillatore sonoro;
36877	X	valore di X compreso tra 128 e 255; seleziona la frequenza del quarto oscillatore dedicato alla generazione di rumore bianco;
36878	V	valore di V compreso tra 0 e 15; seleziona il volume sonoro.

Ponendo nei registri dei particolari valori, e' possibile ottenere su ogni oscillatore tre scale musicali sfasate (tra un oscillatore e l'altro) di un'ottava. I valori da introdurre sono i seguenti:

Valore	Nota	I	Valore	Nota
		I		
135	DO	I	215	SOL
143	DO#	I	217	SOL#
147	RE	I	219	LA
151	RE#	I	221	LA#
159	MI	I	223	SI
163	FA	I	225	DO
167	FA#	I	227	DO#
175	SOL	I	228	RE
179	SOL#	I	229	RE#
183	LA	I	231	MI
187	LA#	I	232	FA
191	SI	I	233	FA#
195	DO	I	235	SOL
199	DO#	I	236	SOL#
201	RE	I	237	LA
203	RE#	I	238	LA#
207	MI	I	239	SI
209	FA	I	240	DO
212	FA#	I	241	DO#

Per chiarire l'uso dei registri del suono forniamo di seguito un programma che permette di suonare un motivo musicale introdotto dal programmatore. Dopo aver assegnato un nome al brano musicale e averlo digitato, e' possibile intervenire su tre parametri: volume, durata delle note ed oscillatore.

PROGRAMMA MUSICALE

```
15 VARIABLE VOLUME <RETURN>

2 VARIABLE OSCILLATORE <RETURN>

1000 VARIABLE DURATA <RETURN>

0 VARIABLE POSIZIONE <RETURN>

: NOTA VOLUME @ 36878 C! POSIZIONE @ C@ DUP 36873
  OSCILLATORE @ + C! <RETURN> DURATA @ 0 DO LOOP 4 0 DO 0
  36874 I + C! LOOP DURATA @ 10 / 0 DO LOOP ; <RETURN>
```

```
: SUONO POSIZIONE ! BEGIN CR ." ?(SHIFT CRSR_)" 3 0 DO KEY
  DUP EMIT LOOP (RETURN) 48 - SWAP 48 - 10 * + SWAP 48 -
  100 * + POSIZIONE @ C! NOTA (RETURN) 1 POSIZIONE +! 0 =
  ?TERMINAL OR UNTIL ; (RETURN)

: SUONA POSIZIONE ! BEGIN NOTA DUP CR . 1 POSIZIONE +! 0 =
  ?TERMINAL OR UNTIL ; (RETURN)

: NOME CR HERE 50 + DUP CONSTANT ; (RETURN)

: SCRIVO DUP 8000 DP ! SUONO POSIZIONE @ 7990 - ALLOT ;
  (RETURN)
```

Dopo aver memorizzato il programma, si operera' nel seguente modo:

1) si assegnera' un nome al motivo musicale da introdurre, digitando la parola NOME ed il nome scelto seguiti da (RETURN);

esempio

```
NOME FRAMARTINO (RETURN)
```

2) si introdurra' la parola SCRIVO seguita da (RETURN);

```
SCRIVO (RETURN)
```

3) il calcolatore sara' ora in attesa dell'introduzione dei numeri corrispondenti alle note: all'apparire di (?) si digiteranno uno di seguito all'altro i numeri scelti;

4) dopo l'ultima nota del brano musicale si digitera' il numero 000 indicante la fine del motivo;

5) il brano potra' ora essere suonato digitandone il nome seguito da SUONA e (RETURN);

```
FRAMARTINO SUONA (RETURN)
```

6) per alterare le variabili VOLUME, OSCILLATORE e DURATA si impieghera' la normale procedura per agire su di esse. Se ad esempio vogliamo suonare il motivetto "Fra Martino" dovremo agire in questo modo:

```
NOME FRAMARTINO <RETURN>

SCRIVO <RETURN>

225 228 231 225 225 228 231 225 231 232 235 231 232
235 235 237 235 232 231 225 235 237 235 232 231 225
228 235 225 228 235 225 000

FRAMARTINO SUONA <RETURN>
```

Il calcolatore suonera' cosi' le note del motivetto introdotto. I motivi musicali vengono automaticamente salvati su cassetta o su disco al momento della registrazione del programma. Sono a disposizione per l'introduzione delle note 3 Kbytes circa di memoria, corrispondenti a ben 3000 note memorizzabili.

DESCRIZIONE DEL PROGRAMMA

Come ogni programma in Forth, anche questo e' strutturato in una serie di parole che definiscono le varie operazioni concatenandosi l'una con l'altra. Diamo qui di seguito la spiegazione di ognuna di queste parole.

NOTA
1) Legge la variabile VOLUME e ne assegna il valore al registro 36878;
2) legge il byte all'indirizzo corrispondente al valore della variabile POSIZIONE, caricandone il contenuto in uno dei 4 registri degli oscillatori;
3) esegue un ciclo DO LOOP da 0 al valore indicato dalla variabile DURATA, determinando la durata della nota stessa;
4) esegue un ciclo DO LOOP per azzerare i 4 oscillatori;
5) con un ultimo ciclo DO LOOP determina la pausa tra una nota e l'altra;

SUONO
1) assegna alla variabile POSIZIONE il valore contenuto nella posizione P1 dello Stack;
2) apre un ciclo BEGIN UNTIL;
3) attende la digitazione del numero di 3 cifre corrispondente alla nota e lo memorizza all'indirizzo indicato dalla variabile POSIZIONE;

4) esegue la parola NOTA ed incrementa il valore della variabile POSIZIONE di 1;
5) chiude il ciclo BEGIN UNTIL. La condizione di uscita e' fornita dalla digitazione o del numero 000 o del tasto <RUN/STOP>.

SUONA 1) assegna alla variabile POSIZIONE il valore contenuto nello Stack;
2) apre un ciclo BEGIN UNTIL;
3) esegue la parola NOTA visualizzando il valore della nota suonata;
4) incrementa il valore della variabile POSIZIONE di 1;
5) chiude il ciclo BEGIN UNTIL. La condizione di uscita e' fornita o dal numero 000 o dalla digitazione del tasto <RUN/STOP>.

NOME 1) con la parola HERE legge l'indirizzo del primo byte libero dopo il vocabolario;
2) definisce una costante assegnandole il valore letto tramite HERE ed il nome del motivo musicale;

SCRIVO 1) posiziona la fine del vocabolario all'indirizzo 8000 agendo sulla variabile DP;
2) esegue la parola SUONO;
3) riposiziona, tramite la parola ALLOT, la fine del vocabolario dopo l'ultima nota del brano musicale introdotto.

Capitolo 14

LA GRAFICA

Grazie alla sua elevata velocita' di elaborazione, il Forth si presta anche per applicazioni di tipo grafico. Nel seguente programma vi diamo un esempio di come si possa realizzare una pagina grafica di 120 per 136 punti, utile per visualizzare, ad esempio, delle funzioni matematiche o dei grafici. Sono inoltre presenti nel programma delle istruzioni che permettono di sovrapporre diverse funzioni, visualizzarle in reverse e di ritornare alla pagina video senza perdere i grafici.

GRAFICA 120*136

```
: VAR VARIABLE ; <RETURN>

0 VAR S <RETURN>

0 VAR X <RETURN>

0 VAR Y <RETURN>

0 VAR V <RETURN>

0 VAR C <RETURN>

: P1 5120 2050 0 FILL 15 36866 C! 34 36867 C! 205 36869 C!
  ; <RETURN>

: P2 ." <CLR>" 256 0 DO ." A" LOOP ; <RETURN>

: P3 255 0 DO I I 4096 + C! LOOP 23 36879 C! ; <RETURN>

: P4 5120 Y @ 8 / 120 * + X @ 8 / 8 * + Y @ + Y @ 8 / 8 * -
  C ! ; <RETURN>

2200 ALLOT <RETURN>
```

```
: P5 2.7 X @ - X @ 8 / 8 * + -1 DO 2 * LOOP 4 / V ! ;
<RETURN>

: P6 V @ C @ C@ OR C @ C! ; <RETURN>

: P7 Y @ 0 < Y @ 135 > OR IF ELSE P4 P5 P6 THEN ; <RETURN>

: P8 X @ 0 < X @ 119 > OR IF ELSE P7 THEN ; <RETURN>

: PLOT S @ 0 = IF P1 P2 P3 1 S ! P8 ELSE P8 THEN ; <RETURN>

: P 22 36866 C! 46 36867 C! 192 36869 C! 27 36879 C! 0 S !
." <CLR>" ; <RETURN>

: G 15 36866 C! 34 36867 C! 205 36869 C! P2 P3 1 S ! ;
<RETURN>

: 0 36879 C@ 27 = IF 23 36879 C! ELSE 27 36879 C! THEN ;
<RETURN>
```

Dopo aver memorizzato il programma saranno disponibili le seguenti nuove parole:

PLOT disegna un punto nella posizione data dalle variabili semplici X e Y. L'origine degli assi X Y e' stata posta nell'angolo in alto a sinistra della pagina grafica. La prima volta che viene utilizzata l'istruzione PLOT, si ha automaticamente il passaggio dalla pagina video a quella grafica.

P Compie un reset ritornando alla pagina video. Eseguendo l'istruzione PLOT dopo la P si provoca la cancellazione della pagina grafica.

G Permette di passare dalla pagina video a quella grafica senza cancellare la rappresentazione delle funzioni precedentemente disegnate. In questo modo e' possibile disegnare una nuova funzione senza cancellare quelle gia' presenti sulla pagina grafica.

0 Esegue il reverse della pagina grafica permettendo di passare da una rappresentazione bianco su nero ad una nero su bianco e viceversa.

ESEMPI DI RAPPRESENTAZIONE DI FUNZIONI MATEMATICHE

```
: F1 120 0 DO I X ! I 50 - DUP * 25 / 5 + Y ! PLOT
  LOOP ; <RETURN>

: F2 120 0 DO I X ! I 30 - I 50 - * 100 / I 80 - * 50
  / 50 + Y ! PLOT LOOP ; <RETURN>

: F3 120 0 DO I X ! 120 I DUP * 100 / I * 50 / - Y !
  PLOT LOOP ; <RETURN>

: F F1 F2 F3 ; <RETURN>
```

Le parole F1 F2 F3 ed F permettono la rappresentazione rispettivamente delle seguenti funzioni matematiche:

F1 $Y=25*(X-50)^2+5$

F2 $Y=(X-30)*(X-50)*(X-80)/5000+50$

F3 $Y=-X^3/5000+120$

F rappresentazione contemporanea di F1 F2 ed F3

Per visualizzare una delle quattro funzioni e' sufficiente digitarne il nome dopo averne, ovviamente, introdotto la definizione.

SCHEMA A BLOCCHI DEL PROGRAMMA

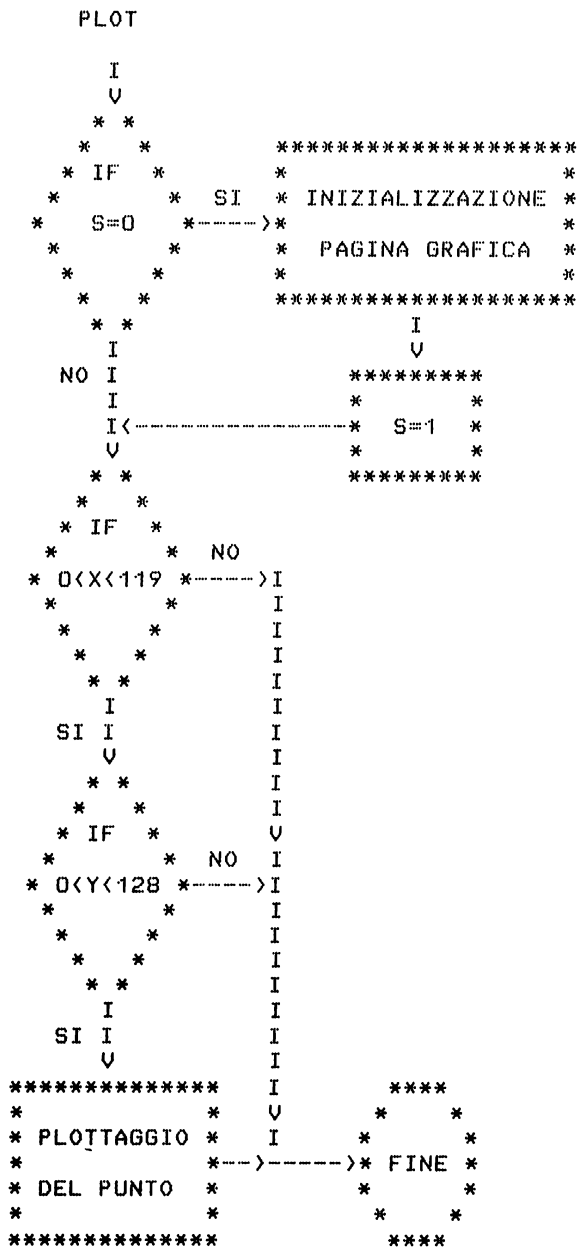


Fig. 35

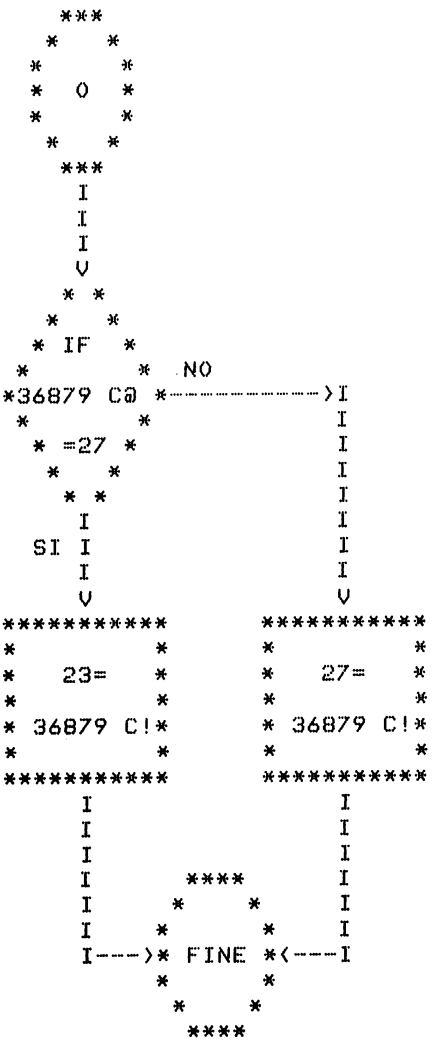


Fig. 36

DESCRIZIONE DEL PROGRAMMA

Trattandosi di un programma di grafica, ricordiamo come sia necessario definire una nuova mappa caratteri in zona RAM. Sara' cosi' possibile ottenere dei disegni creando di volta in volta i caratteri grafici opportuni.
Diamo qui di seguito la funzione di ognuna delle parole utilizzate.

VAR corrisponde alla parola VARIABLE gia' implementata nell'interprete. E' stata definita in quanto, essendo piu' breve a digitarsi, rende piu' agevole la definizione delle variabili.

S E' una variabile semplice utilizzata dall'istruzione PLOT. Il suo valore iniziale e' 0 ma viene modificato in 1 dalle istruzioni PLOT e G e riportato a 0 dalla P. L'istruzione PLOT esegue la reinizializzazione della pagina grafica solo se S=0.

X E' una variabile semplice che assume il valore della ascissa del punto da plottare sul video.

Y E' una variabile semplice che assume il valore dell'ordinata del punto da plottare sul video.

V E' una variabile semplice contenente il valore da utilizzare per alterare opportunamente il byte di indirizzo C della mappa caratteri.

C E' una variabile semplice contenente l'indirizzo del byte della mappa caratteri che deve essere modificato per disegnare il punto.

P1 Prepara la zona di memoria per la nuova mappa caratteri, azzerando i bytes dall'indirizzo 5120 all'indirizzo 7170. Inoltre dimensiona lo schermo in 17 righe per 15 colonne e sposta la mappa caratteri all'indirizzo 5120. Questa zona di memoria e' stata riservata alla mappa caratteri con l'istruzione ALLOT.

P2 Provoca un CLR sul video e scrive nella mappa video la lettera A per 256 volte, in modo da alterare la mappa colore.

P4 Calcola il valore di C in base a quelli di X e Y.

P5 Calcola il valore di V in base a quelli di X e Y.

P6 Compie l'operazione OR tra il contenuto del byte all'indirizzo V ed il valore di C, in modo da sovrapporre il punto determinato da C ad eventuali altri punti gia' presenti.

P7 Esegue le parole P4 P5 P6 solo se e' verificata la condizione: 0<Y<135

P8 Esegue l'istruzione P7 solo se e' verificata la condizione: 0<X<119

PLOT Se S=0 effettua la routine di inizializzazione della pagina grafica tramite P1 P2 P3. Pone poi S=1 ed esegue P8.
Se S=1 esegue direttamente l'istruzione P8.

P Permette di passare dalla pagina grafica a quella video spostando la mappa caratteri all'indirizzo 32768. Ridimensiona il video e da' la combinazione di colori sfondo-bordo originaria. Pone inoltre S=0 e provoca un CLR.

G Inizializza la pagina grafica senza cancellare la mappa caratteri, permettendo cosi' il passaggio alla pagina grafica precedente. Pone inoltre S=1 in modo che, eseguendo successivamente l'istruzione PLOT, non venga reinizializzata la pagina grafica, perdendo cosi' quanto ivi contenuto.

O Se la combinazione colore sfondo-bordo e' 27 la pone uguale a 23. In caso contrario la pone uguale a 27.

Capitolo 15

IL COLORE

All'accensione il video cui e' collegato il VIC 20, presenta la seguente combinazione di colori: bianco per lo sfondo, azzurro per il bordo e blu per il cursore.

Analogamente al funzionamento in Basic, e' possibile modificare il colore del cursore semplicemente premendo il tasto <CTRL> in unione ad uno dei tasti numerici compresi tra 1 e 8. Il cursore assumera' cosi' il colore indicato sulla parte frontale del tasto numerico premuto. Sono pure utilizzabili i comandi di <RVS ON> e <RVS OFF> con l'impiego del tasto <CTRL> contemporaneamente al tasto <9> o <0>.

Per modificare la combinazione di colore sfondo-bordo, si dovra' alterare il contenuto della locazione di memoria 36879. In questo indirizzo si porra' un numero compreso tra 0 e 255, corrispondente ad un numero binario di 8 bits con il seguente significato:

bits 1 2 3 : colore del bordo

bit 4 : visualizzazione in reverse

bits 5 6 7 8 : colore dello sfondo

I numeri immessi nei bits 1 2 3 oppure 5 6 7 8 vengono interpretati dal sistema secondo la seguente tabella:

COLORE	num. decimale	num. binario
NERO	0	0000
BIANCO	1	0001
ROSSO	2	0010
AZZURRO	3	0011
VIOLA	4	0100
VERDE	5	0101
BLU	6	0110
GIALLO	7	0111
ARANCIONE	8	1000
ARANCIO CHIARO	9	1001
ROSA	10	1010
CELESTE	11	1011
VIOLA CHIARO	12	1100
VERDE CHIARO	13	1101
BLU CHIARO	14	1110
GIALLO CHIARO	15	1111

Impiegando la tabella sopra presentata e' possibile calcolare il numero da immettere alla locazione 36879 per ottenere la combinazione di colori desiderata. Piu' semplicemente si puo' utilizzare la seguente tabella riassuntiva:

	BORDO							
SFONDO	NERO	BIANCO	ROSSO	AZZ.	VIOLA	VERDE	BLU	GIALLO
NERO	8	9	10	11	12	13	14	15
BIANCO	24	25	26	27	28	29	30	31
ROSSO	40	41	42	43	44	45	46	47
AZZURRO	56	57	58	59	60	61	62	63
VIOLA	72	73	74	75	76	77	78	79
VERDE	88	89	90	91	92	93	94	95
BLU	104	105	106	107	108	109	110	111
GIALLO	120	121	122	123	124	125	126	127
ARANCIONE	136	137	138	139	140	141	142	143
ARANCIOCHIARO	152	153	154	155	156	157	158	159
ROSA	168	169	170	171	172	173	174	175
CELESTE	184	185	186	187	188	189	190	191
VIOLA CHIARO	200	201	202	203	204	205	206	207
VERDE CHIARO	216	217	218	219	220	221	222	223
BLU CHIARO	232	233	234	235	236	237	238	239
GIALLO CHIARO	248	249	250	251	252	253	254	255

Appendice A

MESSAGGI DI ERRORE

L'interprete Forth segnala gli errori con dei numeri. Nella presente appendice ne diamo l'elenco ed una breve spiegazione.

1 - Stack vuoto

2 - Dizionario vuoto (non usato)

3 - Modo di indirizzamento errato

4 - Nome gia' definito

5 -

6 - Range del disco?

7 - Stack pieno

8 - Errore del disco

9 - Valore o indirizzo non di 8 bit

10 -

11 -

12 - Valore illegale

13 - Troppo ampio

14 -

15 - Datatronic AB (c) 1982 ecc.

16 -

17 - Comando utilizzabile solo in modo differito

18 - Comando utilizzabile solo in modo immediato

19 - Condizione non parificata

20 - Definizione non terminata

21 - Definizione protetta

22 - Impiegare solo in fase di loading

23 - Fuori dallo schermo impiegato

24 - Trasferimento non effettuato

25 - Troppi files

26 - File sempre aperto

27 - File non aperto

28 - File non trovato

29 - Selezione non presente

30 - File non caricato

31 - File non scaricato

32 - Perso nome del file

33 - Illegale numero di selezione

Appendice B

VOCABOLARIO FORTH

Nella descrizione delle parole-chiave del Forth verra' adottata la seguente simbologia:

a	: blocco di istruzioni
addr	: indirizzo del byte
b	: byte di 8 bits
c	: carattere ASCII
cccc	: serie di caratteri ASCII
flag	: condizione logica (0=falso, 1=vero)
n	: numero semplice con segno
nss	: numero semplice senza segno
nd	: numero doppio con segno
ndss	: numero doppio senza segno
ntss	: numero triplo senza segno
pppp	: parola-chiave generica
----	: parola-chiave considerata

Le parole-chiave del vocabolario verranno presentate seguendo l'ordine del codice ASCII.

! n addr ----

memorizza n in addr

!CSP ----

memorizza in CSP l'indirizzo della posizione P1 dello Stack

' ----- pppp addr

fornisce l'indirizzo pfa della parola pppp

((cccc)

permette di inserire, nella definizione di una parola, dei caratteri che non verranno considerati dall'interprete Forth

* n1 n2 ---- n3

esegue il prodotto tra n1 e n2 ottenendo n3

*/ n1 n2 n3 ---- n4

esegue il prodotto n1 n2, lo divide per n3 ed ottiene n4

*/MOD n1 n2 n3 ---- n4 n5

esegue il prodotto n1 n2, lo divide per n3 ed ottiene il quoziente n5 ed il resto n4

+ n1 n2 ---- n3

somma n1 n2 ottenedo n3

+! n addr ----

somma n al valore del byte addr

+- n1 n2 ---- n3

assegna ad n1 il segno di n2 ottenendo n3

+LOOP n1 ----

incrementa di n1 il valore della variabile I del ciclo

, n1 ----

memorizza n1 nella zona vocabolario, incrementando DP di 2

- n1 n2 ---- n3

sottrae n2 ad n1 ottenendo n3

-BCD n1 ---- n2

trasforma n1 in n2 in base al codice BCD

-DUP n1 ---- n1 n1

duplica n1 solo se e' diverso da zero

. n ----

visualizza il numero n

." ." cccc"

visualizza i caratteri cccc

.R n1 n2 ----

visualizza n1 in modo che l'ultima cifra a destra dello stesso sia n2 colonne piu' a destra della posizione precedente del cursore

/ n1 n2 ---- n3

ottiene n3 dividendo n1 per n2

/MOD n1 n2 ---- n3 n4

divide n1 per n2 ottenendo il resto n3 ed il quoziente n4

0 1 2 3 ---- n

sono quattro costanti corrispondenti ai valori 0 1 2 3. Vengono cosi' impiegati, per la memorizzazione del loro valore nella definizione di una parola, solo due bytes anziche' quattro

0< n ---- flag

ottiene il flag 1 se n e' minore di zero

0= n ---- flag

ottiene il flag 1 se n e' uguale a zero

1+ n1 ---- n2

incrementa di 1 n1 ottenendo n2

1+! addr ----

incrementa di uno il valore del numero semplice memorizzato in addr

1- n1 ---- n2

sottrae 1 ad n1 ottenendo n2

1-! addr ----

sottrae 1 al valore del numero semplice memorizzato in addr

2! nd addr ----

memorizza nd in addr

2@ addr ---- nd

carica nello Stack il numero nd contenuto in addr

2+ n1 ---- n2

incrementa di 2 n1, ottendo n2

2- n1 ---- n2

sottrae 2 ad n1, ottenendo n2

2DROP nd1 ----

toglie dallo Stack nd

2DUP nd1 ---- nd1 nd1

duplica nd1

2OVER nd1 nd2 ---- nd1 nd2 nd1

duplica nd1 lasciando inalterato nd2

2ROT nd1 nd2 nd3 ---- nd2 nd3 nd1

ruota le posizioni dei tre numeri

2SWAP nd1 nd2 ---- nd2 nd1

scambia tra loro le posizioni di nd1 e nd2

2VARIABLE nd ---- cccc

definisce una variabile doppia assegnandole il nome cccc

2CONSTANT nd ---- cccc

definisce la costante cccc assegnandole il valore nd

: : cccc a ;

apre la definizione di una nuova parola cccc nel vocabolario Forth

< n1 n2 ---- flag

ottiene il flag 1 se n1 e' minore di n2

= n1 n2 ---- flag

ottiene il flag 1 se n1 e' uguale a n2

> n1 n2 ---- flag

ottiene il flag 1 se n1 e' maggiore di n2

? addr ----

visualizza il numero semplice memorizzato in addr

?COMP ----

trasmette un messaggio di errore se non si sta' immettendo la definizione di una parola

?CSP ----

trasmette un messaggio di errore se il valore di CSP e' diverso dalla posizione dello Stack

?ERROR flag n ----

trasmette il messaggio di errore n se il flag e' 1

?PAIRS n1 n2 ----

trasmette il messaggio di errore 19 se n1 e' diverso da n2

?TERMINAL ---- flag

ottiene il flag 1 solo se e' premuto il tasto <RUN/STOP>

@ addr ---- n

carica nello Stack il numero semplice memorizzato in addr

ABORT ----

Vuota lo Stack e reinizializza il sistema senza cancellare il vocabolario

ABS n ---- nss

fornisce il valore assoluto di n

AGAIN BEGIN a AGAIN

chiude il ciclo BEGIN senza possibilita' di uscita

ALLOT n ----

incrementa di n il valore della variabile DP

AND n1 n2 ---- n3

esegue l'operazione logica AND sui singoli bits dei bytes n1 e n2, ottenendo n3

B. n ----

visualizza n in sistema binario

BASE ---- addr

variabile che determina la base del sistema numerico utilizzato. Inizialmente contiene il valore 10

BEGIN BEGIN a1 pppp

Apre un ciclo che viene chiuso con una delle seguenti parole:
UNTIL
AGAIN
WHILE a2 REPEAT

BL ---- XX

costante contenente il codice ASCII corrispondente ad uno spazio (numero 32)

BLANKS addr n ----

riempie una zona di memoria con il valore 32, a partire dall'indirizzo addr per n bytes

C! b addr ----

memorizza il valore b in addr

C, b ----

memorizza il valore b nella zona vocabolario incrementando DP di uno

C@ addr ---- b

carica nello Stack il valore del byte addr

CFA pfa ---- cfa

trasforma l'indirizzo pfa di una parola nel suo corrispondente cfa

CLOAD ---- f

carica dalla cassetta il vocabolario salvato e fornisce, se non sono stati riscontrati errori nella lettura, un flag 0

CLOSE n ----

chiude il file n precedentemente aperto con OPEN

CMOVE addr1 addr2 n3 ----

sposta il contenuto di un blocco di memoria di n3 bytes, dall'indirizzo addr1 all'indirizzo addr2

COLD ----

provoca un reset completo del sistema

?COMP ----

trasmette un messaggio di errore se non si sta' immettendo la definizione di una parola

CONSTANT n ----- cccc

definisce la costante cccc assegnandole il valore n

2CONSTANT nd ----- cccc

definisce la costante cccc assegnandole il valore nd

CONTEXT variabile contenente l'indirizzo del primo link. Rappresenta cioe' il punto ove l'interprete cerca l'ultima istruzione immessa, ovvero l'inizio del vocabolario

CR ----

posiziona il cursore all'inizio della riga successiva

CSAVE ---- f

salva su cassetta le parole definite nel vocabolario e le variabili, fornendo un flag 1 in caso di errore

CSP ---- addr

variabile utilizzabile per memorizzare l'indirizzo corrispondente alla posizione P1 dello Stack per poter trovare errori di compilazione

!CSP ----

memorizza in CSP l'indirizzo della posizione P1 dello Stack

?CSP ----

trasmette un messaggio di errore se il valore di CSP e' diverso dalla posizione dello Stack

D+ nd1 nd2 ---- nd3

ottiene nd3 sommando nd1 con nd2

D+- nd1 n ---- nd2

ottiene nd2 assegnando a nd1 il segno di n

D. nd ----

visualizza il numero nd

D.R nd n ----

scrive nd in modo che l'ultima cifra a destra dello stesso sia n colonne piu' a destra della posizione precedente del cursore

DABS nd ---- ndss

fornisce il valore assoluto di nd

DECIMAL ----

seleziona il sistema numerico in base decimale assegnando il valore 10 alla variabile BASE

DMINUS nd1 ---- nd2

cambia il segno a nd1 ottenendo nd2

DO n1 n2 ----

apre un ciclo DO LOOP da n1 a n2

DP ---- addr

variabile contenente l'indirizzo del primo byte libero dopo il vocabolario

DPL ---- addr

variabile contenente il numero di cifre a destra della virgola nell'ultimo numero doppio introdotto

DROP n ----

rimuove dallo Stack il numero n

2DROP nd1 ----

toslie dallo Stack nd

DUMP addr1 addr2 ----

visualizza in codice esadecimale e in codice ASCII il contenuto della memoria dall'indirizzo addr1 all'indirizzo addr2

DUP n1 ---- n1 n1

duplica il numero n1

-DUP n1 ---- n1 n1

duplica n1 solo se e' diverso da zero

2DUP nd1 ---- nd1 nd1

duplica nd1

ELSE IF a1 ELSE a2 THEN

segna il termine del blocco di istruzioni a1 (eseguite se il flag precedente l'IF e' 1) e l'inizio del blocco a2 (eseguito se il flag e' 0)

EMIT n ----

visualizza il carattere ASCII corrispondente al numero n

END ----

equivale alla parola-chiave UNTIL

ENDIF ----

equivale alla parola-chiave THEN

ERASE addr n ----

azzera il contenuto di n bytes a partire dall'indirizzo addr

?ERROR flag n

trasmette il messaggio di errore n se il flag e' 1

EXECUTE addr ----

esegue l'istruzione il cui indirizzo cfa e' uguale a addr

EXPECT addr n ----

memorizza in addr una stringa alfanumerica di n caratteri introdotta da tastiera

FENCE ---- addr

variabile contenente l'indirizzo fino a cui non e' possibile operare il FORGET. Puo' essere utilizzato per proteggere delle zone di vocabolario

FILL addr n b ----

memorizza il valore b in n bytes a partire dall'indirizzo addr

FIRST ---- n

costante che fornisce l'indirizzo del primo blocco buffer (1024)

FORGET ---- pppp

cancella il vocabolario dalla parola pppp in poi

H. n ----

visualizza il valore di n in base esadecimale

HERE ---- addr

fornisce l'indirizzo del primo byte libero dopo il vocabolario (corrispondente al valore della variabile DP)

HEX ----

seleziona il sistema numerico in base esadecimale assegnando il valore 16 alla variabile BASE

I ---- n

fornisce il valore dell'indice del ciclo DO LOOP durante la sua esecuzione

I' ---- n

fornisce il valore del limite superiore del ciclo DO LOOP

ID. addr ----

visualizza il nome dell'istruzione il cui indirizzo nfa corrisponde ad addr

IF flag IF a1 THEN

flag IF a1 ELSE a2 THEN

esegue il blocco di istruzioni a1 solo se il flag e' 1 ed il blocco a2 solo se il flag e' 0

LATEST ---- addr

fornisce l'indirizzo nfa dell'ultima parola introdotta

LEAVE ----

provoca l'uscita dal ciclo DO LOOP

LFA addr1 ---- addr2

fornisce l'indirizzo lfa dell'istruzioneil cui pfa e' addr1

LITERAL n ----

memorizza, nella definizione di una parola, il numero n, ottenuto come risultato delle istruzioni a1, nella seguente forma:

a1 LITERAL

LOOP n1 n2 DO a1 LOOP

esegue le istruzioni a1 un numero di volte usuale a n1-n2

+LOOP n1 ----

incrementa di n1 il valore della variabile I del ciclo

M* n1 n2 ---- nd

ottiene nd moltiplicando n1 per n2

M*/ nd1 n nss ---- nd2

moltiplica nd1 per n e divide il risultato per nss ottenendo nd2

M/ nd n1 ---- n2 n3

divide nd per n1 ottenendo il resto n2 ed il quoziente n3

M/MOD ndss1 nss2 ---- nss3 ndss4

divide ndss1 con nss2 ottenendo il resto nss3 ed il quoziente ndss4

MAX n1 n2 ---- n3

fornisce in n3 il maggiore dei due numeri n1 n2

MIN n1 n2 ---- n3

fornisce in n3 il minore dei due numeri n1 n2

MINUS n1 ---- n2

cambia segno ad n1 ottenendo n2

MOD n1 n2 ---- n3

divide n1 per n2 ottenendo il resto n3 (con lo stesso segno di n1)

*/MOD n1 n2 n3 ---- n4 n5

esegue il prodotto n1 n2, lo divide per n3 ed ottiene il quoziente n5 ed il resto n4

/MOD n1 n2 ---- n3 n4

divide n1 per n2 ottenendo il resto n3 ed il quoziente n4

NAME addr n ----

assegna ad un file il nome di lunghezza n memorizzato all'indirizzo addr. Da usarsi prima di CSAVE CLOAD OPEN

NFA addr1 ---- addr2

trasforma l'indirizzo pfa nell'indirizzo nfa corrispondente

OPEN n1 n2 n3 ---- flag

apre il file numero n1 sulla periferica n2 per eseguire l'operazione di tipo n3. Viene dato un flag 0 se il file e' stato aperto correttamente. Prima di eseguire un OPEN e' necessario richiamare il nome assegnato al file con l'istruzione NAME. I valori di n3 corrispondono alle seguenti periferiche:

0 _ video

1 _ registratore

4 _ stampante

8 _ disco

OR n1 n2 ---- n3

ottiene n3 eseguendo l'operazione logica OR tra n1 e n2

OUT ---- addr

variabile indicante il numero totale dei caratteri visualizzati dal sistema

OVER n1 n2 ---- n1 n2 n1

duplica il numero n1

2OVER nd1 nd2 ---- nd1 nd2 nd1

duplica nd1 lasciando inalterato nd2

?PAIRS n1 n2 ----

trasmette il messaggio di errore 19 se n1 e' diverso da n2

PFA addr1 ---- addr2

converte l'indirizzo nfa nel corrispondente pfa

QUERY -----

accetta dalla periferica selezionata un Input alfanumerico fino ad 80 caratteri (chiuso da <RETURN>), che vengono memorizzati nel buffer tastiera. Cio' permette di interrompere l'esecuzione del programma in attesa di un Input che verra' interpretato immediatamente

QUIT -----

fa passare al modo immediato e ritorna il controllo all'operatore

.R n1 n2 -----

visualizza n1 in modo che l'ultima cifra a destra dello stesso sia n2 colonne piu' a destra della posizione precedente del cursore

REPEAT BEGIN a1 flag WHILE a2 REPEAT

chiude il ciclo BEGIN WHILE REPEAT, che esegue le istruzioni a1, esamina il flag (uscendo dal ciclo se e' uguale a 0) ed esegue le istruzioni a2, rincominciando poi il ciclo.

ROT n1 n2 n3 ----- n2 n3 n1

ruota le posizioni di n1 n2 n3

2ROT nd1 nd2 nd3 ----- nd2 nd3 nd1

ruota le posizioni dei tre numeri

S->D n ----- nd

trasforma il numero semplice n nell'equivalente numero doppio nd

SP! -----

vuota lo Stack

SP@ ----- addr

fornisce l'indirizzo della posizione P1 dello Stack

SPACE ----

invia un carattere spazio alla periferica selezionata

SPACES n ----

invia un numero n di caratteri spazio alla periferica selezionata

SMUDGE ----

permette di eseguire il FORGET sull'ultima istruzione introdotta anche se la sua definizione e' incompleta o errata

SP! ----

vuota lo Stack

SWAP n1 n2 ---- n2 n1

scambia le posizioni di n1 e n2

2SWAP nd1 nd2 ---- nd2 nd1

scambia tra loro le posizioni di nd1 e nd2

?TERMINAL ---- flag

ottiene il flag 1 solo se e' premuto il tasto <RUN/STOP>

TIB ---- addr

variabile contenente l'indirizzo dell'inizio del buffer tastiera (80 caratteri). Ha il valore iniziale 256

T* ndss nss ---- ntss

moltiplica ndss per nss ottenendo ntss

T/ ntss nss ---- ndss

divide ntss per nss ottenendo il quoziente ndss

THEN flag IF a1 THEN

flag IF a1 ELSE a2 THEN

esegue il blocco di istruzioni a1 solo se il flag e' 1 ed il blocco a2 solo se il flag e' 0

TOGGLE addr b ----

memorizza in addr il complemento a b del numero precedentemente contenuto in addr

TYPE addr n ----

invia alla periferica selezionata i caratteri ASCII corrispondenti al contenuto di n bytes, a partire dall'indirizzo addr

U* nss1 nss2 ---- ndss3

calcola il valore di ndss3 moltiplicando nss1 per nss2

U. nss ----

visualizza il valore di nss

U/ ndss1 nss2 ---- nss3 nss4

divide ndss1 per nss2 ottenendo il resto nss3 ed il quoziente nss4

UM/* ndss1 nss2 nss3 ---- ndss4

moltiplica ndss1 per nss2, divide il risultato per nss3 ottenendo il risultato ndss4

UNTIL BEGIN a1 flag UNTIL

chiude il ciclo aperto da BEGIN che esegue le istruzioni a1 fino a quando il flag non assume il valore 0

VARIABLE n ---- cccc

definisce una variabile semplice assegnandole il nome cccc ed il valore n

2VARIABLE nd ---- cccc

definisce una variabile doppia assegnandole il nome cccc

VLIST ----

visualizza l'elenco delle parole-chiave memorizzate nel vocabolario. Il tasto <RUN/STOP> interrompe la visualizzazione

WHILE BEGIN a1 flag WHILE a2 REPEAT

fa parte del ciclo BEGIN WHILE REPEAT, che esegue le istruzioni a1, esamina il flag (uscendo dal ciclo se e' uguale a 0) ed esegue le istruzioni a2, rincominciando poi il ciclo.

XOR n1 n2 ---- n3

esegue l'operazione logica EXCLUSIVE-OR tra n1 n2 ottenendo n3

... : pppp a1 a2 a3 ;

permette di eseguire in modo immediato le istruzioni a2, mentre e' in corso la definizione della parola pppp

Appendice C

MAPPA DI MEMORIA

Indirizzi dec.	Indirizzi esa.	Funzione
0	0000	1K RAM area di lavoro per Sistema Operativo e Forth
1023	03FF	
1024	0400	3K RAM espansione contenuta nel cartridge
4095	0FFF	
4096	1000	0.5K RAM mappa schermo
4607	01FF	
4608	0200	3.5K RAM utente
8191	1FFF	
8192	2000	8K RAM/ROM espansione
16383	3FFF	
16384	4000	8K RAM/ROM espansione
24575	5FFF	
24576	6000	8K RAM/ROM espansione
32767	7FFF	
32768	8000	4K ROM mappa caratteri
36863	8FFF	
36864	9000	1K RAM Input/Output 0
37877	93FF	
37888	9400	0.5K RAM mappa colore
38399	95FF	
38400	9600	0.5K RAM
38911	97FF	

38912	9800	1K RAM Input/Output 2
38935	98FF	
38936	9C00	1K RAM Input/Output 3
40959	9FFF	
40960	AFFF	8K ROM interprete Forth
49151	BFFF	
49512	C000	8K ROM interprete Basic
57343	DFFF	
57344	E000	8K ROM sistema operativo
65535	FFFF	

Appendice D

COLORI SFONDO-BORDO

Per modificare la combinazione di colori sfondo-bordo, si dovra' alterare il contenuto della locazione di memoria 36879. In questo indirizzo si porra' un numero compreso tra 0 e 255 secondo la seguente tabella:

	BORDO							
SFONDO	NERO	BIANCO	ROSSO	AZZ.	VIOLA	VERDE	BLU	GIALLO
NERO	8	9	10	11	12	13	14	15
BIANCO	24	25	26	27	28	29	30	31
ROSSO	40	41	42	43	44	45	46	47
AZZURRO	56	57	58	59	60	61	62	63
VIOLA	72	73	74	75	76	77	78	79
VERDE	88	89	90	91	92	93	94	95
BLU	104	105	106	107	108	109	110	111
GIALLO	120	121	122	123	124	125	126	127
ARANCIONE	136	137	138	139	140	141	142	143
ARANCIOCHIARO	152	153	154	155	156	157	158	159
ROSA	168	169	170	171	172	173	174	175
CELESTE	184	185	186	187	188	189	190	191
VIOLA CHIARO	200	201	202	203	204	205	206	207
VERDE CHIARO	216	217	218	219	220	221	222	223
BLU CHIARO	232	233	234	235	236	237	238	239
GIALLO CHIARO	248	249	250	251	252	253	254	255

Appendice E

IL SUONO

Indirizzi delle locazioni di memoria da modificare per la generazione dei suoni:

indirizzi	valore	funzione
36874	X	valore di X compreso tra 128 e 255; seleziona la frequenza del primo oscillatore sonoro;
36875	X	valore di X compreso tra 128 e 255; seleziona la frequenza del secondo oscillatore sonoro;
36876	X	valore di X compreso tra 128 e 255; seleziona la frequenza del terzo oscillatore sonoro;
36877	X	valore di X compreso tra 128 e 255; seleziona la frequenza del quarto oscillatore dedicato alla generazione di rumore bianco;
36878	V	valore di V compreso tra 0 e 15; seleziona il volume sonoro;

Valori da introdurre per ottenere le note musicali su tre ottave:

Valore	Nota	I	Valore	Nota
		I		
135	DO	I	215	SOL
143	DO#	I	217	SOL#
147	RE	I	219	LA
151	RE#	I	221	LA#
159	MI	I	223	SI
163	FA	I	225	DO
167	FA#	I	227	DO#
175	SOL	I	228	RE
179	SOL#	I	229	RE#
183	LA	I	231	MI
187	LA#	I	232	FA
191	SI	I	233	FA#
195	DO	I	235	SOL
199	DO#	I	236	SOL#
201	RE	I	237	LA
203	RE#	I	238	LA#
207	MI	I	239	SI
209	FA	I	240	DO
212	FA#	I	241	DO#

Appendice F

PROGRAMMI

Forniamo nella prima parte di quest'appendice i listati dei principali programmi del libro; nella seconda parte presentiamo invece alcuni nuovi programmi con una breve spiegazione del loro impiego.

PRIMA PARTE

PROGRAMMA IN$

```
100 VARIABLE LM <RETURN>

0 VARIABLE T <RETURN>

0 VARIABLE P <RETURN>

: L? P @ T @ < IF ELSE DROP 13 DUP P @ C! 1 P +! THEN ;
  <RETURN>

: IN$ CR DUP P ! LM @ + T ! BEGIN KEY DUP EMIT DUP P @ ! 1
  P +! L? 13 = UNTIL ; <RETURN>
```

PROGRAMMA OUT$

```
: F? P @ T @ < IF ELSE DROP 13 1 P +! THEN ; <RETURN>

: OUT$ CR DUP P ! LM @ + T ! BEGIN P @ C@ DUP  EMIT 1 P +!
  F? 13 = UNTIL ; <RETURN>
```

```
PROGRAMMA =$


0 VARIABLE $1 <RETURN>

0 VARIABLE $2 <RETURN>

: E? 2DUP = $2 @ T @ < AND 1 SWAP 2SWAP 13 = SWAP 13 = OR 1
  SWAP - AND - ; <RETURN>

: SUB $1 @ C@ $2 @ C@ E? 1 $1 +! 1 $2 +! ; <RETURN>

: =$ $1 ! DUP $2 ! LM @ + T ! BEGIN SUB UNTIL $1 @ 1- C@ $2
  @ 1- C@ = ; <RETURN>


PROGRAMMA L$


: L$ DUP $1 ! 1- T ! BEGIN 1 T +! T @ C@ 13 = UNTIL T @ $1
  @ - ; <RETURN>


PROGRAMMA R$


0 VARIABLE T1 <RETURN>

0 VARIABLE T2 <RETURN>

0 VARIABLE LL <RETURN>

: S4 1 $1 +! 1 $2 +! ; <RETURN>

: S3 BEGIN $1 @ C@ DUP $2 @ C@ = SWAP 13 = 1 SWAP - AND
  WHILE S4 REPEAT ; <RETURN>

: S2 T1 @ $1 ! T2 @ $2 ! S3 $1 @ C@ 13 = ; <RETURN>

: S1 DUP T2 ! DUP L$ + LL ! T1 ! ; <RETURN>

: R$ S1 BEGIN S2 1 SWAP - T2 @ LL @ < AND WHILE 1 T2 +!
  REPEAT S2 ; <RETURN>
```

PROGRAMMA MUSICALE

```
15 VARIABLE VOLUME <RETURN>

2 VARIABLE OSCILLATORE <RETURN>

1000 VARIABLE DURATA <RETURN>

0 VARIABLE POSIZIONE <RETURN>

: NOTA VOLUME @ 36878 C! POSIZIONE @ C@ DUP 36873
  OSCILLATORE @ + C! <RETURN> DURATA @ 0 DO LOOP 4 0 DO 0
  36874 I + C! LOOP DURATA @ 10 / 0 DO LOOP ; <RETURN>

: SUONO POSIZIONE ! BEGIN CR ." ?<SHIFT CRSR→)" 3 0 DO KEY
  DUP EMIT LOOP <RETURN> 48 - SWAP 48 - 10 * + SWAP 48 -
  100 * + POSIZIONE @ C! NOTA <RETURN> 1 POSIZIONE +! 0 =
  ?TERMINAL OR UNTIL ; <RETURN>

: SUONA POSIZIONE ! BEGIN NOTA DUP CR . 1 POSIZIONE +! 0 =
  ?TERMINAL OR UNTIL ; <RETURN>

: NOME CR HERE 50 + DUP CONSTANT ; <RETURN>

: SCRIVO DUP 8000 DP ! SUONO POSIZIONE @ 7990 - ALLOT ;
  <RETURN>
```

PROGRAMMA GRAFICA 120*136

```
: VAR VARIABLE ; <RETURN>

0 VAR S <RETURN>

0 VAR X <RETURN>

0 VAR Y <RETURN>

0 VAR V <RETURN>

0 VAR C <RETURN>

: P1 5120 2050 0 FILL 15 36866 C! 34 36867 C! 205 36869 C!
  ; <RETURN>

: P2 ." <CLR>" 256 0 DO ." A" LOOP ; <RETURN>
```

```
: P3 255 0 DO I I 4096 + C! LOOP 23 36879 C! ; <RETURN>

: P4 5120 Y @ 8 / 120 * + X @ 8 / 8 * + Y @ + Y @ 8 / 8 * -
  C ! ; <RETURN>

2100 ALLOT <RETURN>

: P5 2 7 X @ - X @ 8 / 8 * + -1 DO 2 * LOOP 4 / V ! ;
  <RETURN>

: P6 V @ C @ C@ OR C @ C! ; <RETURN>

: P7 Y @ 0 < Y @ 135 > OR IF ELSE P4 P5 P6 THEN ; <RETURN>

: P8 X @ 0 < X @ 119 > OR IF ELSE P7 THEN ; <RETURN>

: PLOT S @ 0 = IF P1 P2 P3 1 S ! P8 ELSE P8 THEN ; <RETURN>

: P 22 36866 C! 46 36867 C! 192 36869 C! 27 36879 C! 0 S !
  ." <CLR>" ; <RETURN>

: G 15 36866 C! 34 36867 C! 205 36869 C! P2 P3 1 S ! ;
  <RETURN>

: 0 36879 C@ 27 = IF 23 36879 C! ELSE 27 36879 C! THEN ;
  <RETURN>
```

SECONDA PARTE

PROGRAMMA RSAVE

Semplifica la procedura necessaria per salvare un programma assegnandogli un nome. Va impiegato digitando RSAVE <RETURN> ed il nome assegnato al programma seguito da <RETURN> (la lunghezza del nome non deve superare i 20 caratteri)

```
: RSAVE HERE 20 2DUP EXPECT NAME CSAVE ; <RETURN>
```

PROGRAMMA RLOAD

Permette di caricare un programma di nome assegnato, digitando RLOAD <RETURN> seguito dal nome del programma e da <RETURN>.

```
: RLOAD HERE 20 2DUP EXPECT NAME CLOAD ; <RETURN>
```

PROGRAMMA STACK

Visualizza i numeri semplici con segno contenuti nelle singole posizioni dello Stack. Se nello Stack non e' memorizzato alcun valore appare la scritta:

VUOTO

Si impiega digitando S seguito da <RETURN>.

```
: S ." <SHIFT CRSR_>TACK " 116 SP@ 2+ 2DUP = IF ." VUOTO "
  2DROP ELSE DUP ROT ROT <RETURN> DO CR I OVER - 2 / 1+ ."
  P" . I ? <RETURN> BEGIN ?TERMINAL 1 SWAP - UNTIL 2 +LOOP
  DROP THEN ; <RETURN>
```

PROGRAMMA ELEVAMENTO A POTENZA

Digitando <RETURN>, il numero semplice con segno contenuto nella posizione P2 dello Stack viene elevato all'esponente contenuto in P1. Nelle posizioni P1 P2 verra' memorizzato il numero doppio con segno risultante.

```
:   DUP 0 = IF 2DROP 1. ELSE DUP 1 = IF DROP 0 ELSE OVER
  SWAP 0 ROT ROT <RETURN> 1 DO DUP 2SWAP ROT T* DROP ROT
  LOOP DROP THEN THEN ; <RETURN>
```

PROGRAMMA RADICE QUADRATA DI UN NUMERO SEMPLICE

Digitando SQR <RETURN> viene calcolata la radice quadrata del numero semplice senza segno contenuto nella posizione P1 dello Stack. Il risultato viene memorizzato nella posizione P1 dello Stack come numero semplice.

```
: SQR 0 BEGIN 1+ 2DUP DUP * < UNTIL SWAP DROP 1- ; <RETURN>
```

PROGRAMMA RADICE QUADRATA DI UN NUMERO DOPPIO

Digitando DSQR <RETURN> viene calcolata la radice quadrata del numero doppio senza segno (minore di 603.979.775) contenuto nelle posizioni P1 P2 dello Stack. Il risultato viene memorizzato nella posizione P1 dello Stack come numero semplice.

```
: DSQR 2 1 BEGIN 2OVER 2OVER DROP DUP M* DMINUS D+ SWAP
  DROP 0 < <RETURN> IF 2 / 2DUP - ELSE 2 * 2DUP + THEN ROT
  DROP SWAP <RETURN> DUP 0 = UNTIL 2SWAP 2DROP DROP 1- ;
  <RETURN>
```

PROGRAMMA FATTORIALE

Digitando F e <RETURN> viene calcolato il fattoriale del numero semplice contenuto nella posizione P1 dello Stack. Il numero doppio risultante viene memorizzato nelle posizioni P1 P2 dello Stack.

```
: F DUP 0 SWAP BEGIN DUP 1 > WHILE 1- DUP 2SWAP ROT T* DROP
  ROT REPEAT DROP ; <RETURN>
```

Appendice G

FORTH DEL CBM 64

La versione del Forth sviluppata dalla Datatronic per il CBM 64 e' sostanzialmente analoga a quella per il VIC 20 cui si e' fatto riferimento nel libro. Le differenze tra le due versioni sono originate quasi esclusivamente dal diverso hardware del CBM 64 rispetto al VIC 20 (maggior quantita' di memoria disponibile, visualizzazione su 40 colonne anziche' su 22, differente gestione della grafica e del suono, ecc.).

Le principali differenze tra le due versioni sono:

1) Possibilita' di interrompere l'esecuzione di un programma premendo contemporaneamente i due tasti <RUN/STOP> e <RESTORE>;

2) Comparsa, dopo l'interruzione di un programma, della scritta <RESTARTED>;

3) Diverso ordine di visualizzazione delle parole-chiave del vocabolario Forth in seguito alla digitazione dell'istruzione VLIST;

4) Inizio memoria RAM disponibile all'indirizzo 2289;

5) 30 Kbytes di memoria RAM disponibili per la programmazione.

Tutti gli esempi ed i programmi presenti nel libro fino a pagina 90 girano perfettamente anche sul CBM 64. I successivi programmi di grafica e musica, sfruttando delle peculiari caratteristiche del VIC 20, dovranno essere modificati per il loro impiego sul CBM 64.

INDICE ANALITICO

VLIST	11 12 13 14 128
WHILE	68 85 128
XOR	42 128
	128

INDICE DELLE FIGURE

L'ASSEMBLER

GUIDA ALLA PROGRAMMAZIONE IN ASSEMBLER Z80 SUL PICO COMPUTER
di Dante Del Corso

Il libro
È una guida introduttiva alla programmazione assembler attraverso una progressione di esercizi. Il calcolatore usato è il Picocomputer, che impiega il microprocessore Z80 di cui non viene volutamente fornita una descrizione generale.
I programmi riportati possono essere facilmente adattati ad altri sistemi Z80 o 8080. Di ogni programma viene fornito il listato completo e quindi non occorre disporre di assemblatori o altri supporti di sviluppo, oltre il Pico stesso o piastra equivalente.
Sommario
Sistema PICOCOMPUTER - Esercizi - Tabella delle istruzioni Z80 - Standard Mubus - Tastiera e display: tecniche di interfacciamento - Scheda CPU: criteri di progetto e descrizione dell'hardware - Scheda CPU: montaggio e collaudo - Scheda CPU: estensioni - Programma monitor - Interfaccia cassette - Tecniche di interfacciamento su Mubus.

Cod. 330D **pag. 138** **L. 9.000**

PROGRAMMARE IN ASSEMBLER
di Alain Pinaud

Il libro
Una schiera sempre più vasta di hobbisti e/o utenti di personal computer vorrebbero avvicinarsi alla programmazione in assembler, ma esita perché lo ritiene terribilmente complesso e necessitante di lunghi studi.
È possibile invece, con questo libro in poco tempo e con semplicità apprendere quei principi base validi per qualsiasi microprocessore, a 8, 16, 32 o 64 bit. Poiché, però bisognava far riferimento ad un assembler esistente, si è scelto quello dello Z80, sia perché tra i più diffusi, sia perché dotato del set di istruzioni più ampio nella sua categoria.
Sommario
Definizione e richiami di nozioni di base - Introduzione all'assembler - Istruzioni di un assembler tipo Z80 - Pseudoistruzioni e macroistruzioni - Tecnica pratica dell'assembler - Il software di supporto all'assemblatore - Relazioni con i linguaggi evoluti - La matematica dell'informatica - Correzione degli esercizi - Il codice ASCII - Il set di istruzioni dello Z80.

Cod. 329 **pag. 160** **L. 10.000**

GRUPPO EDITORIALE JACKSON
Divisione Libri

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista.

Scrive, suona, gioca, entusiasma

Gaetano Marano

66 PROGRAMMI PER ZX 81

E ZX80 CON NUOVA ROM + HARDWARE

Per le sue qualità e il suo modestissimo prezzo lo ZX 81 della Sinclair è il computer più venduto nel mondo.
Oggi, sempre con una modestissima spesa, si può imparare a sfruttare questo eccezionale strumento al limite delle sue capacità. Basta scorrere questo libro per scoprire quante cose lo ZX 81 può fare con l'aggiunta di alcuni semplici ed economici componenti. Ad esempio, tramite un semplice circuito musicale può riprodurre 50 note su 4 ottave e, sempre grazie a una modifica hardware da poche migliaia di lire, lo ZX 81 diventa anche l'unico computer in grado di conferire effetti sonori ai giochi inseriti tra i suoi programmi. Ma non è tutto. Un'altra novità di quest'opera, preziosa anche per chi possiede lo ZX 80 con ROM, è il regalo di alcune tastiere disegnate da sovrapporre a quella sensitiva dell'apparecchio, per ricavarne altre, speciali funzioni.

136 pagine. Lire 12.000 Codice 520 D

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista

GRUPPO EDITORIALE JACKSON

Una guida pratica, preziosa, aggiornata

General Electric

LA SOPPRESSIONE DEI TRANSITORI DI TENSIONE

Un libro che riassume i risultati delle pluriennali ricerche effettuate da una delle massime industrie mondiali sulle cause, gli effetti, la frequenza dei sovraccarichi di tensione derivanti dai disturbi atmosferici o da altri motivi. Un'opera eminentemente pratica che si propone di dare ai tecnici un contributo fattivo alla soluzione di questo annoso problema, anche attraverso l'indicazione della vasta gamma di dispositivi di protezione che la G.E. ha messo a punto sulla scorta dei suoi studi e delle esperienze.

216 pagine. Lire 12.000 Codice 611 A

GRUPPO EDITORIALE JACKSON

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista

IL BASIC E LA GESTIONE DEI FILE

Il libro si rivolge in modo particolare a chi già conosce il Basic e desidera poter realizzare programmi che prevedano l'uso di file residenti su disco. Dopo aver preso in esame, utilizzando numerosi esempi pratici, le particolarità del Microsoft, si passa alla descrizione delle istruzioni necessarie ad una corretta gestione dei file su disco, sia ad accesso diretto che sequenziale. Una terza parte del libro è infine interamente dedicata alla esposizione dei metodi pratici per l'uso dei file ad accesso diretto e dei data base.

Cod. 515H **L. 11.000 Pagg. 164**

50 ESERCIZI IN BASIC

Una raccolta completa e progressiva di esercizi riguardanti matematica, gestione, ricerca operativa, gioco e statistica. Ciascun esercizio proposto comporta l'enunciazione e l'analisi del problema, la risoluzione mediante flow-chart e commenti, così come un programma che implementa la soluzione, illustrato da semplici esempi rappresentativi.
Questo metodo mette in grado il lettore di verificare passo passo le sue conoscenze e il livello di apprendimento raggiunto.

Cod. 521A **L. 13.000 Pagg. 208**

75 PROGRAMMI IN BASIC PER IL VOSTRO COMPUTER

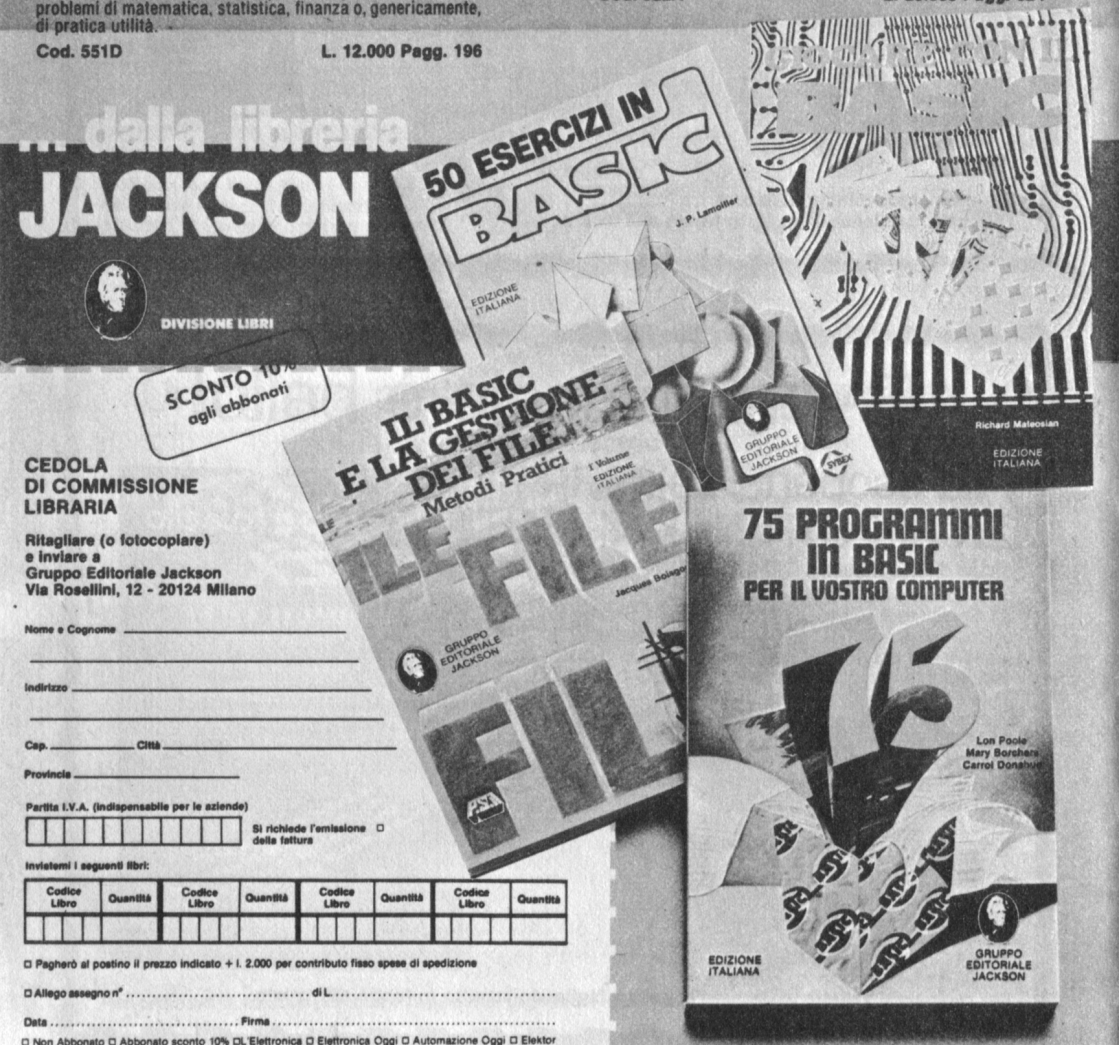
Il volume raccoglie ben 75 programmi originali scritti in un Basic generico, utilizzabili, salvo poche e semplici modifiche, sulla maggior parte dei personal computer in commercio, a cassetta come a floppy disk. Per ciascuno, dopo una descrizione introduttiva, viene fornito il listing e un campione di esecuzione. Così come sono, i programmi proposti (tutti verificati), costituiscono un valido ausilio per chiunque debba risolvere problemi di matematica, statistica, finanza o, genericamente, di pratica utilità.

Cod. 551D **L. 12.000 Pagg. 196**

GIOCARE IN BASIC

Nei giochi, il lettore può ritrovare tutte quelle situazioni reali di programmazione che gli saranno indispensabili nella comprensione e realizzazione di qualsiasi applicazione interattiva del proprio computer, anche le più sofisticate. Questo senza annoiarsi, ma entrando da subito all'interno della materia per imparare a comprendere il BASIC, il proprio computer e i computer in genere.

Cod. 522A **L. 20.000 Pagg. 324**



SCONTO 10% agli abbonati

CEDOLA DI COMMISSIONE LIBRARIA

Ritagliare (o fotocopiare) e inviare a Gruppo Editoriale Jackson Via Rosellini, 12 - 20124 Milano

Nome e Cognome _____

_____

Indirizzo _____

_____

Cap. _____ Città _____

Provincia _____

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura ☐

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + l. 2.000 per contributo fisso spese di spedizione

☐ Allego assegno n° ..... di L. .....

Data ..... Firma .....

☐ Non Abbonato ☐ Abbonato sconto 10% ☐ L'Elettronica ☐ Elettronica Oggi ☐ Automazione Oggi ☐ Elektor ☐ Informatica Oggi ☐ Computerworld ☐ Bit ☐ Personal Software ☐ Strumenti Musicali ☐ Videogiochi

PROGRAMMI DI MATEMATICA E STATISTICA

Leggendo questo libro il lettore potrà formarsi quella logica di base indispensabile per la risoluzione di problemi di matematica e statistica.
Ad ogni programma viene preposta un'esposizione schematica del metodo numerico e delle tecniche di programmazione utilizzate, il diagramma a blocchi relativo all'algoritmo, il listato (anch'esso ottenuto da calcolatore) in cui tra l'altro vengono specificati il tempo e la quantità di memoria impiegati.

Cod. 522D **L. 16.000 Pagg. 228**

INTRODUZIONE AL PASCAL

Il volume, incentrato su numerosissimi esempi che verificano costantemente l'apprendimento del lettore, insegna a conoscere, capire ed usare tutte le particolarità e i vantaggi di questo linguaggio. Nel corso della trattazione vengono ampiamente utilizzate le tecniche di programmazione strutturata, come pure tecniche particolari, quali il trattamento dei file, l'utilizzazione della recursività e il trattamento grafico.

Cod. 516A **L. 30.000 Pagg. 484**

COMPUTER GRAFICA

Si può dire che la computer grafica si pone nel contesto più generale del trattamento dell'informazione, avendo individuato nell'immagine un contenuto informativo che è possibile elaborare.
Quest'opera, con il suo rigore informativo e scientifico, si pone come fondamentale nel carente panorama italiano; inoltre le informazioni e gli spunti contenuti nel testo contribuiranno certamente alla divulgazione ed alla formazione di idee nuove e feconde.

Cod. 519P **L. 29.000 Pagg. 174**

APPLE II - Guida all'uso

Se possedete un Apple e volete conoscerlo a fondo, se volete comprarlo, o se semplicemente volete imparare la sua programmazione, troverete in questo libro, tutte le risposte, comprese alcune vere "primizie" che vi occorrono per una perfetta operatività del sistema. Conoscerete i vari componenti del sistema e come usarli al meglio. Verrete guidati alla programmazione in BASIC e a usare le caratteristiche grafiche e sonore del sistema. Imparerete a memorizzare su disco sia programmi che archivi dati, come ad inserire un programma scritto in assembler in uno scritto in BASIC.
E poi ancora, tutte le istruzioni e funzioni BASIC e ben 12 appendici veramente basilari.

Cod. 331P **L. 26.000 Pagg. 400**

CEDOLA DI COMMISSIONE LIBRARIA

Ritagliare (o fotocopiare) e inviare a
Gruppo Editoriale Jackson Via Rosellini, 12 - 20124 Milano

Nome e Cognome _____

Indirizzo _____

Cap. _____ Città _____ Provincia _____

Codice Fiscale (indispensabile per le aziende)

Inviatemi i seguenti libri:

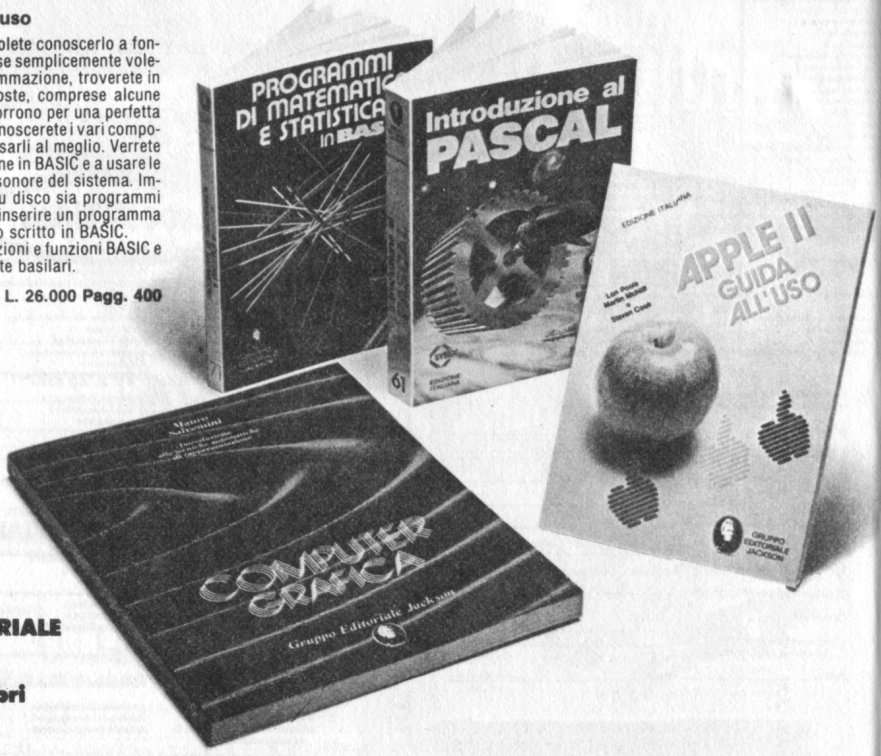
Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione

☐ Allego assegno n° ..... di L. .....

Data ..... Firma .....

... dalla libreria JACKSON



GRUPPO EDITORIALE JACKSON

Divisione Libri

CEDOLA DI COMMISSIONE LIBRARIA

Da inviare a Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

Nome Cognome

Indirizzo

Cap. **Città** **Provincia**

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura ☐

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione.

☐ Allego assegno n° ..... di L. .....

☐ Non abbonato ☐ Abbonato sconto 20% ☐ l'Elettronica ☐ Elettronica Oggi ☐ Automazione Oggi ☐ Elektor
☐ Informatica Oggi ☐ Computerworld ☐ Bit ☐ Personal Software ☐ Strumenti Musicali ☐ Videogiochi

Data ..... Firma .....

CEDOLA DI COMMISSIONE LIBRARIA

Da inviare a Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

Nome Cognome

Indirizzo

Cap. **Città** **Provincia**

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura ☐

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione.

☐ Allego assegno n° ..... di L. .....

☐ Non abbonato ☐ Abbonato sconto 20% ☐ l'Elettronica ☐ Elettronica Oggi ☐ Automazione Oggi ☐ Elektor
☐ Informatica Oggi ☐ Computerworld ☐ Bit ☐ Personal Software ☐ Strumenti Musicali ☐ Videogiochi

Data ..... Firma .....

L. 11.000 Cod. 527 B ISBN 88-7056-141-0

Il libro si pone come obiettivo principale quello di spiegare la programmazione in Forth, linguaggio che, dopo essersi affermato in campo scientifico ed industriale, sta ora diffondendosi anche a livello di personal computer.
In modo agile e scorrevole vengono spiegati al lettore i vantaggi del Forth rispetto ad altri linguaggi: l'elevata velocità di elaborazione (inferiore in molti casi solo del 10 per cento rispetto al linguaggio-macchina) e la compattezza di programmazione, caratteristiche che rendono il Forth estremamente adatto all'impiego in applicazioni in tempo reale oppure in campo gestionale.
Pur svolgendo un discorso generale sul Forth, nel libro viene fatto riferimento ad una implementazione del linguaggio disponibile sui computer Commodore VIC 20 e CBM 64. Questa versione del Forth, peraltro molto simile a quella disponibile per altri personal computer quali, ad esempio, lo ZX SPECTRUM, presenta delle differenze minime rispetto alla versione standard, volute per meglio sfruttare le capacità di suono e colore proprie del VIC 20 e CBM 64.
Il libro si sviluppa su due piani: il primo costituisce il "manuale" di apprendimento delle regole e dei principi del Forth (primi dieci Capitoli ed Appendici A, B); il secondo, passando da un discorso generale ad uno specifico, spiega l'uso del Forth con il VIC 20 e CBM 64.
Arricchiscono infine il libro ben quattordici programmi, dal trattamento di stringhe al Programma Musicale, di cui viene data una dettagliata spiegazione del funzionamento ed il diagramma di flusso.

86

FORTH

per VIC 20 e CBM 64

Giacomino Baisini
Gio. Federico Baglioni

GRUPPO EDITORIALE JACKSON

MUSEO DEL COMPUTER
Fondazione ONLUS
Sede legale
Via Costantino Perazzi 22
28100 NOVARA
Tel 0321 1856032
www.museodelcomputer.org
info@museodelcomputer.org

C.F. 94064520037
Registro P.G. 237

Documento redatto da Alberto Rubinelli in data 06-01-2012 Revisione 1.4