FORTH PER SPECTRUM

per chi desidera scoprire la potenzialità del FORTH, il linguaggio ad alto livello enormemente più veloce del BASIC.

di DON THOMASSON

edizioni jce

FORTH PER SPECTRUM

per chi desidera scoprire la potenzialità del FORTH, il linguaggio ad alto livello enormemente più veloce del BASIC.

di DON THOMASSON

TRADUZIONE DI
LUCA BRIGATTI

edizioni JCE

Via dei Lavoratori, 124
CINISELLO BALSAMO (MI)

Pubblicato nel Regno Unito da:
Melbourne House (editori) Ltd.,
Church Yard
Tring, Herthfordshire HP23 5LU.
ISBN 0 86161 142 x

Pubblicato in Australia da:
Melbourne House (Australia) Pty. Ltd.,
Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria, 3205.

Prima edizione italiana : Agosto 1984

Stampato in Italia da
Gemm Grafica S.r.l.
Via Magretti - Paderno Dugnano (MI)

INDICE

INTRODUZIONE

Il FORTH e' il prodotto della mente di un uomo che si chiama Charles Moore. Dopo aver provato vari aspetti del linguaggio per diversi anni, inizio' a sintetizzarli verso la fine degli anni sessanta, ci volle pero' ancora del tempo prima che ne emergesse una versione completamente coerente. Considerando la sua invenzione come la quarta generazione dei linguaggi per computers, ma costretto dalla sua attrezzatura a nomi di cinque lettere, scelse di far cadere la "u" e chiamo' la sua idea "FORTH".

Da allora, il linguaggio fu sviluppato in diverse direzioni, le due versioni principali divennero FORTH79 e fig-FORTH. Il prefisso "fig" si riferisce al FORTH Interest Group di San Carlos, California che ha cercato di rendere accessibile il linguaggio al maggior numero possibile di utenti.

Tale numero oggi si estende a utenti di una gran quantita' di microcomputers, ma ogni implementazione e' leggermente differente dalle altre, la definizione originale del fig-FORTH ammette ampie possibilita' per la scelta individuale delle carateristiche. questo libro e' basato specificamente sul FORTH Spectrum della Abersoft, che e' forse la versione piu' semplice e completa disponibile. La maggior parte di quanto verra' detto e' comunque applicabile ad altre versioni del fig-FORTH e, dove altre versioni omettono alcune delle parole descritte, sara' possibile implementarle per ottenere l'Abersoft standard, anche se alcune caratteristiche saranno piu' difficili da ottenere.

Il FORTH79 e' un'altra faccenda. Usa parole diverse e da' ad alcune parole altri singificati; e mentre il concetto di base e' essenzialmente lo stesso del fig-FORTH, l' implementazione e' piuttosto differente. Tuttavia, la flessibilita' e' una caratteristica di entrambi i tipi di FORTH, e questo puo' consentire di levigare alcune delle discrepanze tramite l'acquisizione di nuove parole e la revisione delle definizioni.

Un grosso problema che i principianti del FORTH devono affrontare riguarda le dimensioni del "vocabolario", che e' la lista di parole predefinite. In genere e' tabulata nell'ordine di rappresentazione delle parole secondo il codice ASCII, ma quando il vocabolario viene visualizzato in risposta all'istruzione VLIST (seguito da return) le parole sono in un ordine completamente diverso. In ogni caso, trovare una parola per fare una determinata operazione non e' facile. L'approccio adottato qui e' di raggruppare le parole di uso piu' frequente a seconda del genere di azioni che queste esplicano, in modo che un dato tipo di funzione possa essere localizzato velocemente. In aggiunta, l'Appendice A fornisce una definizione completa delle parole standard nella forma di un sommario compresso del "dizionario", l'area di memoria nelle quali sono definite le parole.

Comunque, sarebbe sbagliato aspettarsi di poter scrivere un programma FORTH immediatamente. L'approccio migliore e' di imparare le parole operative a gruppi, sperimentando con queste finche' non si acquisisce una profonda conoscenza del sistema.

Poi potrete iniziare a definire parole nuove per conto vostro con grande confidenza e, piu' tardi, a definire interi programmi.

Questo approccio progressivo e' possibile poiche' il FORTH puo' essere usato a diversi livelli. Esso interpretera' singole parole esistenti o una serie di parole in modo "diretto" che puo' essere paragonato al modo diretto del BASIC. D'altro canto, compilera' nuove parole , creando per loro nuovo spazio nel dizionario, e queste parole possono essere usate per la compilazione di altre definizioni. Alla fine, una singola parola chiamera' un programma completo.

Gli inserimenti nel dizionario sono fatti in modo tale che le routines associate ad ogni parola possano essere trovate molto velocemente. Dove un interprete BASIC deve eseguire una scansione su una tabella per trovare il punto d'ingresso richiesto, il FORTH memorizza il punto d'ingresso direttamente, e non e' richiesta alcuna scansione. Questo rende il FORTH molto piu' veloce del BASIC, il che e' forse il suo maggior pregio. Un altro pregio e' l'economia nell'utilizzo della memoria.

Per apprezzare questi vantaggi, dovete essere preparati ad aiutare il linguaggio facendo cose che avreste fatto automaticamente usando il BASIC. Dovete pensare un po' di piu', tenendo un occhio su cio' che il programma sta facendo. In compenso, il FORTH vi dara' una flessibilita' limitata soltanto dalla vostra immaginazione.

Per cominciare

L'Abersoft impiega circa 70 secondi per essere caricato; quindi dovrebbe apparire sullo schermo questa intestazione:

```
48k SPECTRUM fig-FORTH (versione)
(c) Abersoft: 1983
```

Questo indica che il vostro Spectrum e' diventato una macchina FORTH.

Il primo segno del cambiamento e' che il cursore lampeggiante mostra una "C" anziche' una "L". Poiche' la maggior parte delle parole standard del FORTH sono in caratteri maiuscoli, e' piu' conveniente lavorare in "modo maiuscolo", comunque potete sempre selezionare il modo "L" nel solito modo, usando CAPS SHIFT e 2.

Noterete che il lieve beep indicante la pressione dei tasti e' sparito. Se ritorna, siete passati dal FORTH al BASIC per qualche motivo, ma e' poosibile tornare al FORTH con GOTO 3. Potete tornare deliberatamente al BASIC battendo MON e ENTER ma scoprirete che e' possibile solo l'esecuzione diretta perche' non c'e' posto per fare aggiunte al breve programma BASIC gia' presente.

I tasti non produrranno piu' parole BASIC, che non sono necessarie in FORTH. Dovrete battere tutto lettera per lettera. Non c'e' un modo esteso ma troverete che alcuni simboli normalmente ottenuti in questo modo, piu' qualche altro, possono essere ottenuti usando SYMBOL SHIFT e certi altri tasti, es.:

```
Il tasto "Y" da' [
Il tasto "U" da' ]
Il tasto "A" da' ~
```

Il tasto "S" da' ;
Il tasto "D" da' \
Il tasto "F" da' {
Il tasto "G" da' }

La grafica e' accessibile nel solito modo.

Una volta persa l'abitudine di usare il tasto "K" per LIST, che e' una parola FORTH, non incontrerete seri problemi.

Sono state mantenute un certo numero di funzioni famigliari, come spiegato in dettaglio nel paragafo "Caratteristiche particolari dello Spectrum", ma scoprirete che non hanno esattamente la stessa forma che nel BASIC, per esempio, i parametri devono precedere l'istruzione, non seguirla.

Un'importante differenza concerne il BREAK. In alcune circostanze, premendo CAPS SHIFT e SPACE si interrompera' comunque il programma, ma viene fornita un'alternativa da CAPS SHIFT e 1. Questo, tuttavia, funzionera' solo se avete scritto il programma prevedendo questo input. Se non avete previsto il BREAK, potreste scoprire che il programma e' entrato in un loop infinito senza possibilita' di uscita. Dettagli sul modo di evitarlo vengono forniti nel paragrafo sulle diramazioni e ricorsioni ma non e' necessario che vi preoccupiate per il momento.

Se decidete di fare un altro tipo di interruzione, caricando un altro tipo di programma, dovete rilocare la RAMTOP, altrimenti otterreste il messaggio "Out of Memory". Il sistema piu' facile e' spegnere il computer e riaccenderlo, cosi' che vengano inizializzati i soliti parametri.

Se vi interessano queste faccende, l'area iniziale occupata dal programma FORTH dopo il caricamento va da 5E06 a 8159, con riservato dello spazio di lavoro da CB40 in poi. Lo spazio intermedio e' disponibile per le vostre parole FORTH, sotto forma di estensione del "dizionario" che e' la definizione fondamentale del FORTH. Il dizionario aumenta verso l'alto, e lo stack del calcolatore aumenta verso il basso. Se volete sapere quanto spazio rimane fra di loro, battere:

FREE . ENTER

Verra' mostrato il numero di bytes liberi.

La ZX Printer puo' funzionare col FORTH. Accendetala con:

1 LINK ENTER

Spegnetela con:

0 LINK ENTER

A causa del fatto che il FORTH e' per molti versi differente dalla magioranza degli altri linguaggi, il prossimo paragrafo esaminera' le sue caratteristiche generali. Se le conoscete gia', potete tranquillamente saltarlo.

Nozioni Essenziali sul FORTH

Usato in modo diretto, il FORTH lavora come un calcolatore che opera in notazione inversa (Reverse Polish Notation). I dati che battete vengono mostrati sullo schermo e memorizzati in un

buffer' di 80 caratteri, che conterra' 2,5 linee di dati . Quando il buffer e' pieno, o se premete ENTER indicando che l'input e' completo, il sistema comincia a scandire il buffer, fermandosi al primo spazio che trova. Il dato che e' stato scandito viene quindi confrontato con la lista di parole nel dizionario, che puo' essere vista tramite:

VLIST ENTER

Potere fermare il listing premendo BREAK (CAPS SHIFT/1), dato che la lista occupa piu' spazio dell'area dello schermo.

Se il dato corrisponde a una parola definita in FORTH, viene eseguita l'azione associata a quella parola . Altrimenti, viene controllato se il dato e' un numero valido. Se lo e', il numero viene posto nello stack del calcolatore. Se non e' ne' una parola FORTH ne' un numero valido, allora viene visualizzato l'errore 0, con la parola scorretta.

Poiche' si usa la notazione inversa, con i dati memorizzati in uno stack, le parole FORTH agiscono sugli ultimi numeri inseriti. Provare questo:

4 5 * .

Lasciate uno spazio fra ogni elemento dell'input, ricordate che lo "spazio" e' il "delimitatore" standard che segna l'inizio e la fine di una parola o di un numero. In risposta, verra' visualizzato il numero 20. Il sistema di funzionamento e' il seguente: il FORTH prende i dati dal primo spazio, in questo caso il numero 4. Poiche' si tratta di un numero e non di una parola FORTH, questo viene memorizzato nello stack. Il procedimento e' ripetuto per il numero 5 e il risultato viene memorizzato sopra il 4. Quindi il FORTH incontra il "*". Poiche' e' una parola FORTH e ci troviamo nel modo diretto, questa viene eseguita. Il FORTH prende l'ultimo numero (5) dallo stack e quello immediatamente sotto (4) e li moltiplica, mettendo il risultato nello stack. La prossima parola nel buffer e' il punto ".". Questo fa apparire il risultato.

Se inseriamo quanto segue:

4 5 * 23 + .

Fino al "*" tutto e' come prima, con il risultato (20) nello stack. Il FORTH continua a leggere la linea e memorizza il 23 nello stack sopra il 20, poi va a leggere la parola seguente. Il (+) e' una parola FORTH; somma i due numeri in cima allo stack, 23 e 20, e memorizza il risultato, 43, ancora nello stack. Il "punto", di nuovo, visualizza il risultato.

Le azioni delle parole FORTH, in questi inputs sono state spiegate in un modo relativamente semplice che dovrebbe bastare per il momento, ma e' evidente che sono necessarie definizioni esatte, che verranno fornite in seguito.

Non e' necessario scrivere un set completo di numeri e di parole su una stessa linea, ognuna di queste puo' andare su una linea diversa, seguita da ENTER, visto che il sistema di input tratta solo un elemento per volta. Se inserite troppi caratteri, sovrassaturando il buffer, otterrete probabilmente un messaggio d'errore, dato che l'ultima parola inserita puo' non essere completa, ma i primi elementi della linea saranno trattati normalmente e, siccome il messaggio d'errore identifica la

1 Area di memoria destinata a contenere temporaneamente informazioni in ingresso o in uscita (N.d.T.)

parola a cui si riferisce, saprete da dove riprendere l'inserimento.

Non tutte le parole mostrate da VLIST possono essere usate in modo diretto, e un tentativo di usarle produrra' l'errore 17. Queste parole possono essere usate nel modo compilazione, che stabilisce nuovi inserimenti nel dizionario definendo nuove parole FORTH.

Ecco un esempio:

```
: TEST1 * + . ;
```

I due punti informano il sistema che tutte le parole e i numeri fino al seguente punto e virgola devono essere compilate e il risultato identificato con la parola TEST1.

Ora inserite quanto segue:

```
23 4 5 TEST1 ;
```

Il risultato e' 43, come nell'esempio di prima. TEST1 esegue la moltiplicazione, l'addizione, e la funzione di scrittura che prima avevamo inserito come parole separate. Sarebbe una buona idea simulare lo stack su carta per essere sicuri di avere i valori e gli operatori nell'ordine corretto. Se lo fate per l'operazione di cui sopra, vedrete come il 23 venga prima.

Potreste definire un'altra parola:

```
: TEST2 23 4 5 TEST1 ;
```

Inserendo TEST2 verra' mostrato 43. Cio' non ha molto senso, ma serve ad illustrare come funziona la gerarchia FORTH. Alla fine, viene chiamato un programma completo inserendo una singola parola. Questa parola definisce una lista di altre parole da eseguirsi a turno, e alcune o tutte le parole della lista possono definire ulteriori liste.

E' interessante confrontare cio' con l'azione del BASIC, che usa linee numerate contenenti istruzioni che specificano l'azione da eseguirsi. La numerazione viene usata per indicare l'ordine nel quale le istruzioni vanno eseguite. Un programma BASIC ben strutturato, tuttavia, puo' essere visto come un certo numero di blocchi funzionali, ognuno composto di un certo numero di linee, anche se alcuni programmi contengono talmente tante istruzioni su una linea che una linea puo' rappresentare un blocco. Cio' rende il programma difficile da interpretare dal listato.

Il FORTH, d'altro canto, da' ad ogni blocco funzionale un nome che e' associato ad una lista di nomi che definiscono le funzioni che devono essere espletate dal blocco. Gruppi di blocchi sono analogamente associati a ulteriori nomi e liste, finche' un solo nome e lista porta tutti i blocchi in un unico programma chiamato da quel singolo nome.

Chiaramente, questo non e' un processo cui si puo' accedere troppo a caso. Siccome un programma BASIC puo' essere modificato con le funzioni di editing, e' possibile a volte creare semplici routines procedendo a caso e correggendo gli errori nonappena diventano ovvi. Con il FORTH, e' consigliabile una certa pianificazione. Questo puo' imporre una disciplina tediosa se preferite l'approccio piu' casuale permesso dal BASIC, ma la

velocita' d'esecuzione del risultato rende conveniente lo sforzo.

Un problema importante per coloro che si avvicinano al FORTH per la prima volta, e' semplicemente il numero di parole disponibili . L'unica soluzione e' imparare il vocabolario in sezioni, espandendo gradualmente la vostra comprensione sul raggio d'azione che potete trattare. Su questa base e' impostato il resto di questo libro. Se la spiegazione dettagliata di qualche parola sembra troppo complessa, lasciatela perdere e tornateci piu' tardi, accettando che la parola funziona come descritto. Scoprirete che potrete molto presto cominciare a capire quello che succede, anche per funzioni molto complesse.

Un'altro problema per i principianti, sta nel fatto che tutti i dati numerici vengono memorizzati in formato binario. Non molti anni or sono, il sistema binario era considerato incomprensibile per chiunque non fosse un esperto di computers. Dopo una lezione su tale soggetto, un uomo si alzo', scarabocchio' una serie di uno e di zero sulla lavagna, dicendo che nessuno avrebbe potuto ricavare un senso da cio'. Come scrisse l'ultimo zero, una flebile voce dall'auditorium disse: "trecentottantaquattro". L'uomo sgrano' gli occhi e si mise laboriosamente a controllare il valore di quell' "incomprensibile" numero binario, per scoprire che la conversione era corretta. Il ragazzino che aveva fatto la conversione apparve sorpreso e spiego' che tutto cio' che aveva fatto era stato di prendere la prima cifra e, quando veniva aggiunta un'altra cifra, aveva raddoppiato quello che aveva e sommato un'altra cifra. Per esempio:

```
1        1
1        3
0        6
0       12
0       24
0       48
0       96
0      192
0      384
```

Il numero scritto alla lavagna era 110000000, non certo il piu' difficile dei numeri da convertire, ma il principio e' utile da ricordare e si applica ad ogni base numerica.

Per esempio, in base 3, il numero 112020 si converte cosi':

```
1        1
1        4
2       14
0       42
2      128
0      384
```

Il FORTH esegue questa operazione per qualsiasi base desideriate selezionare, non solo in input ma anche in output. Questo puo' generare confusione se si sbaglia a valutare cio' che accade, cosi' lo studio del vocabolario iniziera' con una occhiata al modo in cui i numeri vengono inseriti, memorizzati ed emessi.

CAPITOLO I:

Il modo diretto

Questo capitolo tratta di quelle parole FORTH che possono essere inserite per l'esecuzione immediata. Alcune versioni del BASIC possono essere usate in questo modo che viene allora denominato modo "diretto" o "calcolatore". In un capitolo successivo verra' esaminato l'uso della compilazione per la creazione di nuove parole, il che si dimostrera' piu' utile per operazioni complesse. Tuttavia, benche' lavorare in modo diretto possa sembrare piu' lento e tedioso, da' un'idea migliore di cio' che accade realmente.

RAPPRESENTAZIONE DEI NUMERI

La chiave delle operazioni del FORTH e' lo "stack". Per coloro che non hanno famigliarita' con il termine, uno stack[2] e' un buffer che puo' essere paragonato allo spillone che usano le cassiere per conservare gli scontrini. Se mettiamo gli scontrini nello spillone quando ci vengono consegnati, il primo ad uscire sara' l'ultimo ad essere stato inserito. Il FORTH usa due stacks, uno per i calcoli ed uno per contenere gli indirizzi associati. Questi saranno denominati Stack del Calcolatore (o, spesso, solamente Stack) e Return Stack rispettivamente.

Nel FORTH Spectrum, lo stack dello Z80 viene usato per lo Stack del Calcolatore. Lo stack Z80 tratta dati in forma di parole a 16 bit, anche quando non e' coinvolto piu' di un carattere ASCII, cio' si addice perfettamente al FORTH. I computers che usano altri processori possono avere stacks che trattano parole a 8 bit, ma devono memorizzare due parole successive per ogni trasferimento allo stack. In generale, tranne che per funzioni molto specializzate, il FORTH rimane inalterato dalle caratteristica della macchina.

Un altro punto e' che il FORTH non fa distinzioni fra i tipi di dati, il che occasionalmente puo' avere conseguenze sconcertanti. Il significato delle parole di dati dipende dal modo in cui sono trattate. Per esempio, battete 40000 e poi ENTER. Quindi battete un punto, seguito ancora da ENTER. Il punto mostra il numero che si trova in cima allo stack, che e', ovviamente, l'ultimo numero inserito. Il numero mostrato e' -25536 e non 40000. Da dove e' sbucato?

Numeri con segno e senza segno

Un numero binario a 16 bit puo' avere 65536 valori differenti. Nella rappresentazione "binaria pura" questi valori variano fra 0 e 65535, avendo il bit n valore 2^n, dove il bit meno significativo e' il bit 0. Se, d'altro canto, adottiamo la rappresentazione "in complemento a 2", il valore del bit piu' significativo e' -2^15 anziche' 2^15. Il resto dei bits puo'

2 Letteralmente "catasta". (N.d.T.)

rappresentare valori da 0 a 32767. Quando il bit 15 e' 0, tale e' il campo di valori del numero completo. Se il bit 15 e' 1, allora viene sottratto 32768 (2^15), avendosi un campo di valori da -32768 a -1. Siccome il numero e' sempre positivo se il bit 15 e' a 0 e negativo quando il bit 15 e' a 1, questo bit e' detto bit del "segno" (sign bit).

Qui e' necessario un avvertimento riguardo al numero -32768. Questo si comporta in modo un poco bizzarro, rimanendo immutato dalla negazione. Riguardo a questo, puo' essere paragonato allo zero. E' meglio evitarlo, lavorando con un campo di valori di ±32767.

Il FORTH vi consente di scegliere fra la rappresentazione binaria pura o quella in complemento a 2. Il semplice punto, mostrera' un numero sulla base del complemento a 2, mentre U. assumera' che il numero sia in binario puro. Siccome il numero 40000 e' fuori dal campo di valori del complemento a 2, e' stato memorizzato come binario puro. Usando il punto, il numero viene interpretato come se fosse in complemento a 2. Sia 40000 che -25536 sono rappresentati dal numero binario 1001110001000000. Notate che questi possono raggiungere il valore di 65536.

Numeri doppi

Il campo di valori per un numero a 16 bit e' piuttosto limitato e cosi' il FORTH prevede l'uso di numeri a 32 bit, che vengono trattati come due successivi numeri nello stack, con i 16 bit superiori piu' vicini alla cima dello stack. Possono essere rappresentati 4.294.867.206 valori differenti, e in complemento a 2 si ha un campo di valori di 2.147.483.647. Il numero anomalo e', in questo caso, -2.147.483.648.

E' possibile inserire un numero doppio mettendo un punto decimale da qualche parte nel numero. La posizione del punto non e' direttamente rilevante, ma il numero di cifre alla sua destra viene annotata nella variabile DPL per un possibile riferimento futuro.

Un fatto da ricordare e' che un numero fuori dal campo di valori 32767, inserito senza un punto decimale, puo' causare problemi. La routine di input interpeta sempre un input numerico come numero doppio, ma se non ci sono punti decimali la parte superiore del risultato viene eliminata. Questo puo' dare occasionalmente risultati mistificanti.

Un numero doppio puo' essere visualizzato dalla parola D. , che assume che si tratti di un numero doppio e in complemento a 2.

Basi numeriche

Quando non altrimenti specificato, il FORTH accetta inputs numerici e genera outputs numerici intendendo che si tratti di numeri decimali, convertendo i numeri nella e dalla rappresentazione binaria interna. Se altrimenti istruito, esso lavorera' con qualsiasi base numerica si desideri, entro limiti ragionevoli. La parola HEX chiama la notazione esadecimale, la parola DECIMAL ristabilisce la notazione decimale. E questo, e' solo l'inizio.

La parola ! dice al FORTH di memorizzare un numero nella seconda posizione dello stack in una locazione definita dalla cima dello stack. La frase n BASE ! memorizzera' dunque il numero n nella variabile BASE, che determina la base numerica da usarsi in input e in output. HEX equivale a 16 BASE ! e DECIMAL equivale a 10 BASE ! , mentre per lavorare in binario si scrive 2 BASE ! . Queste istruzioni presuppongono che sia operativo il decimale, se e' stato selezionato il binario, 10 BASE ! non fa niente, visto che in binario 10 ha valore due! Dovete fare 1010 BASE ! .

Per cifre con valore maggiore di 9, le lettere da A a Z servono a estendere il campo di valori delle cifre, sarebbe cosi' possibile lavorare in base 35; oltre tale valore, verranno usati caratteri strani.

La parola @ (SYMBOL SHIFT + 2) dice al FORTH di prendere il contenuto della locazione definita dalla cima dello stack e di mettere il risultato nello stack, cosi', potreste pensare che la base corrente possa essere determinata da BASE @ . , ma il risultato e' sempre 10. Riuscite a capire perche'? Piu' avanti c'e' un aiuto.

Usando la notazione esadecimale, e' preferibile adottare la convenzione di numeri senza segno in "binario puro", poiche' i numeri esadecimali negadivi creano confusione.

Numeri Formattati

Una delle virtu' del FORTH e' la capacita' di visualizzare i numeri in formato molto preciso. L'esempio piu' semplice chiama in causa la parola .R per numeri singoli con segno, D.R per numeri doppi, e U.R per numeri singoli senza segno.

Queste parole prendono il numero in cima allo stack che definisce un "campo" di quel numero di posizioni di caratteri. Il prossimo numero dello stack viene visualizzato alla destra del campo. Se un output numerico normale aggiunge uno spazio dopo il numero, queste funzioni non lo fanno. E' quindi relativamente facile costruire tabulazioni con i numeri "giustificati a destra", cioe' con le cifre piu' a destra incolonnate. Altre possibilita' di questo genere emergeranno a tempo debito.

Nel frattempo, abbiamo cominciato definendo dieci parole e frasi FORTH.

Per comodita', queste possono essere riassunte come segue:

TOS (Top Of Stack) significa "Cima dello stack", 2OS (2nd On Stack) significa "il secondo nello stack" e cosi' via.

.	Rimuove TOS e lo visualizza come numero singolo con segno.
U.	Rimuove TOS e lo visualizza come numero singolo senza segno.
D.	Rimuove TOS, 2OS e li visualizza come numero doppio con segno.
n BASE !	Stabilisce la notazione in base n.
HEX	Stabilisce la notazione esadecimale.
DECIMAL	Stabilisce la notazione decimale.

BASE @ .	Legge la base corrente.
.R	Rimuove TOS, 2OS. Visualizza 2OS come singolo numero con segno alla destra di un campo di TOS caratteeri.
U.R	Rimuove TOS, 2OS. Visualizza 2OS come singolo numero senza segno alla destra di un campo di TOS caratteri.
D.R	Rimuove TOS, 2OS, 3OS e visualizza 2OS, 3OS come numero doppio con segno alla destra di un campo di TOS caratteri.

FUNZIONI ARITMETICHE PRIMITIVE

Le parole FORTH sono costruite sulla base di un certo numero di blocchi di codici macchina. Altre parole sono create combinando o modificando queste parole "primitive", ma l'azione di queste ultime puo' essere compresa solo se prima si esaminano le primitive.

Potreste essere sorpresi dal fatto che ci siano solo otto primitive aritmetiche. Piu' avanti vedremo che queste formano la base di altre diciassette derivate.

E' importante ricordare che le parole FORTH agiscono su numeri o altri dati che sono gia' nello stack. Per addizionare 7 e 8, dobbiamo fare:

```
7 8 +
```

Esaminando quello che e' successo, cominciamo battendo i caratteri e gli spazi, che sono memorizzati nel buffer di input terminale. La routine INTERPRET prende, a turno, i dati delimitati dagli spazi. Vede che 7 e' un numero valido, avendolo prima considerato come una possibile parola FORTH, e il numero 7 va nello stack. 8 e' trattato in modo simile, e va anch'esso nello stack. Il numero 8 e' "TOS" e il 7 e' "2OS". Il segno piu', tuttavia, e' riconosciuta come parola FORTH, e viene eseguita. Prende le due cifre in cima allo stack, le somma e mette il risultato di nuovo nello stack. Possiamo adesso visualizzare il risultato usando il punto oppure "U.".

Incidentalmente, non c'era bisogno di porre tutti e tre i caratteri sulla stessa linea. L'effetto sarebbe stato lo stesso se avessimo inserito:

```
7
8
+
```

Questo perche' il FORTH opera su ogni parola o numero a turno. Premendo ENTER, invitiamo semplicemente INTERPRET a esaminare quello che e' stato inserito nel buffer di input terminale (Terminal Input Buffer, abbreviato in TIB). Se volete sapere dove si trova il TIB, HEX TIB @ U. ve lo dira'. Fa tutto parte del servizio FORTH!

Anche il segno meno e' una parola FORTH, rimuove i primi due numeri dello stack, sottrae TOS da 2OS e pone il risultato di nuovo in TOS.

Questo tipo di calcolo e' noto come notazione inversa, e poiche' e' stato usato in qualche calcolatore, puo' non essere del tutto sconosciuto. Esso evita ambiguita', giacche' gli operatori agiscono in un modo definito esattamente su numeri definiti esattamente. Dove la notazione aritmetica normale necessita di parentesi e di regole di priorita' per determinare la sua interpretazione, la notazione inversa non lascia posto a dubbi.

D'altro canto, richiede che i numeri giusti si trovino nella posizione giusta per poter essere elaborati nel momento giusto, e cio' puo' richiedere una certa prudenza attraverso una preventiva pianificazione. Una scorsa alle definizioni del dizionario in Appendice A rivelera' qualche esempio interessante.

La prossima "primitiva" e' U* , che prende TOS e 2OS e mette il loro prodotto nello stack sotto forma di numero doppio. Poi abbiamo U/MOD che prende i primi tre numeri nello stack, tratta il secondo e il terzo come numero doppio da dividere per il primo numero. Il resto viene messo nello stack come singolo numero e il quoziente viene messo anch'esso nello stack sopra il resto.

Mettere il resto nello stack puo' sembrare inutile, ma non e' cosi' per un'ottima ragione. Supponete di avere nello stack un numero doppio che rappresenta dei secondi. Aggiungendo 60 U/MOD il numero verra' diviso per 60 e il quoziente dara' il numero dei minuti, il resto saranno i secondi che rimangono. Provate:

```
200. 60 U/MOD . .
```

Il punto decimale dopo 200 e' necessario per stabilire che questi e' un numero doppio perche' U/MOD possa funzionare. Possiamo usare i punti per gli outputs poiche' i risultati saranno minori di 32767 e non si fara' confusione con i segni. Provate ora:

```
36484. 60 U/MOD 0 60 U/MOD . . .
```

L'inserimento dello zero e' necessario per fare, della parola singola risultante dalla prima operazione, una parola doppia in modo che il secondo U/MOD possa lavorare. Le tre cifre rappresentano ore, minuti e secondi.

Questo puo' essere visto come un calcolo in base 60, e la routine per gli output decimali lavora in una base simile a quella mostrata sopra, tranne che 60 e' sostituito da 10.

Ci sono volte in cui un numero singolo va convertito in un numero doppio, magari per poter usare U/MOD o un altro operatore simile. Se U/MOD viene usato con un numero singolo, questi pigliera' qualsiasi cosa si trovi nello stack e lavorera' su di essa. Una volta ancora, potrebbero esserci strani risultati.

Se il numero singolo e' senza segno, possiamo estenderlo mettendo uno zero nello stack a formare la meta' superiore del numero. Se il numero singolo e' con segno, e vogliamo conservare il segno, dobbiamo usare S->D , che mette uno zero nello stack se il numero singolo e' positivo o FFFFH se il numero singolo e' negativo. Questa vien detta "propagazione del bit segno".

Per sommare fra di loro due numeri doppi, va usato D+ . Esso rimuove quattro numeri dallo stack, li tratta come due numeri

doppi, li somma e mette il risultato nello stack come numero doppio.

Le ultime due primitive eseguono la negazione, MINUS agisce sui numeri singoli e DMINUS sui numeri doppi. La loro azione e' di sottrarre il numero da zero. Per tutti gli scopi, il numero sembra rimanere nello stesso punto dello stack, infatti viene prelevato e rimesso ancora nello stack.

Queste otto primitive possono sembrare alquanto inadeguate, ma prima di esaminare le diciassette derivate, dobbiamo considerare i manipolatori dello stack, senza i quali il FORTH sarebbe veramente molto limitato.

Gli operatori descritti in questo paragrafo possono essere cosi' riassunti:

+	Rimuove TOS, 2OS. Pone la loro somma nello stack.
-	Rimuove TOS, 2OS. Pone 2OS-TOS nello stack.
U*	Rimuove TOS, 2OS. Pone il loro prodotto nello stack sotto forma di numero doppio.
U/MOD	Rimuove TOS, 2OS, 3OS. Tratta 2OS e 3OS come numero doppio, lo divide per TOS. Mette il resto nello stack e poi il quoziente. Tutti i numeri sono considerati senza segno.
D+	Rimuove TOS, 2OS, 3OS, 4OS. Considerandoli due numeri doppi, pone il numero doppio risultante dalla loro somma nello stack.
S->D	Estende il segno di un numero singolo a formare un numero doppio. Il numero originale diventa 2OS, l'estensione TOS.
MINUS	Nega il numero singolo in TOS.
DMINUS	Nega il numero doppio in TOS e 2OS.

I MANIPOLATORI DELLO STACK

E' evidente che le operazioni matematiche e le altre operazioni creeranno una certa confusione nello stack per poter portare il dato richiesto nell'esatta posizione. Il che non e' un problema grave quando il numero di dati nello stack e' piccolo, ma una routine che sara' descritta piu' avanti coinvolge fino a 72 numeri nello stack contemporaneamente, e questo e' un problema piu' complesso.

Una delle parole piu' sfruttate in FORTH e' DUP ,che duplica il TOS aggiungendo una copia del TOS originale. E' essenziale quando il TOS viene controllato, poiche' il controllo distrugge il numero se non era stato duplicato. Una variante e' -DUP che duplica il TOS solo se non e' zero. E' utile dove non si abbia piu' bisogno del TOS quando questo raggiunge il valore zero. C'e' anche 2DUP, che duplica TOS e 2OS. E' usato per duplicare numeri doppi, ma puo' essere usato anche con numeri singoli. Poniamo che si voglia calcolare (4+3)*3+4 . Possiamo usare:

```
                Stack
4 3             4 3
2DUP            4 3 4 3
+               4 3 7
U*              4 21 0
DROP            4 21
+               25
```

U* genera un numero doppio, cosi' scartiamo la parte superiore del numero a 16 bit con DROP , che scarta TOS. 2DROP scarta TOS e 2OS.

Poi abbiamo SWAP, che scambia TOS e 2OS e 2SWAP che scambia TOS/2OS con 3OS/4OS.

OVER e' molto utile. Aggiunge un nuovo TOS con una copia del precedente 2OS. 2OVER aggiunge un nuovo TOS e 2OS che sono la copia dei precedenti 3OS e 4OS.

Finalmente, in questo gruppo, ROT ruota i primi tre numeri dello stack, portando 3OS a TOS, 2OS e TOS a 2OS.

Quando comincerete a scrivere programmi di ampia portata, scoprirete che e' molto utile - se non essenziale - scrivere tabelle come quella riportata piu' sotto, per poter seguire i movimenti dello stack.

Facendo cio', vi renderete presto conto che non c'e' modo di effettuare i cambiamenti necessari per portare numeri sepolti profondamente nello stack, in cima a questo, ma scoprirete che il problema e' semplificato quando impareremo a compilare definizioni di nuove parole.

A causa di questa limitazione, conviene spesso introdurre numeri nel corso dei calcoli. Questi possono essere inclusi in iputs, o possono - come vedremo a tempo debito - essere presi da costanti e variabili. Siccome cio' occupa ulteriore memoria, viene considerato antieconomico dai puristi del FORTH ma non ha senso sforzarsi troppo per raggiungere la perfezione.

I manipolatori che sono stati menzionati possono essere definiti come segue:

	Stack prima	**Stack dopo**
DUP	a	a a
2DUP	a b	a b a b
DROP	a	
2DROP	a b	
SWAP	a b	b a
OVER	a b	a b a
2OVER	a b c d	a b c d a b
ROT	a b c	b c a

Ricordate che la cima dello stack e' il numero all'estrema destra . Vengono mostrati solo i dati rilevanti. Ci possono essere altri dati "dietro" a questi, cioe' piu' profondi nello stack e a sinistra di quelli mostrati. Questi rimangono inalterati dall'azione dei manipolatori.

ALTRI MANIPOLATORI

A questo punto e' necessario menzionare alcuni manipolatori dello stack e del display che non rientrano convenientemente in altre categorie.

In primo luogo ce ne sono tre che non sono accessibili in modo diretto ma che vengono usati per produrre alcune funzioni aritmetiche derivate. Abbiamo gia' fatto riferimento al Return Stack. Qualche volta e' conveniente sottrarre la cima dello stack del calcolatore dall'azione immediata, magari per permettere l'accesso a dati piu' sotto nello stack. Queste parole facilitano il compito permettendo alla cima dello stack del calcolatore di essere trasferita temporaneamente nel Return Stack.

>R	rimuove il TOS e trasferisce il dato in TORS (Top of Return Stack, cioe' cima del Return Stack).
R<	esegue la funzione opposta.
R	copia il TORS in TOS, senza alterare TORS.

Queste parole vanno usate con prudenza, il Return Stack viene riportato al suo stato originale prima che sia completata una definizione, o si faccia riferimento al contenuto del Return Stack per altri scopi, come nel controllo dei cicli ricorsivi. (E' stata presa una cantonata di questo genere nella definizione di 2OVER nei primi nastri Abersoft. Se scoprite che HEX 7F4E @ U. da' 6173, avete l'errore, che puo' essere corretto con HEX 6188 DUP 7F4E ! 7F50 !).

I manipolatori dello schermo sono piuttosto ovvi. CR chiama una nuova linea (Newline), CLS cancella lo schermo. SPACE emette uno spazio, mentre n SPACES emette n spazi.

Nel modo diretto, *." stringa"* scrive i caratteri stringa delimitati dagli apici. Corrisponde al BASIC PRINT "XXXXX".

ALTRE FUNZIONI ARITMETICHE

A questo punto, armati dei manipolatori dello stack, possiamo vedere come sono state create le altre diciassette funzioni aritmetiche. Dobbiamo comunque dare prima uno sguardo agli operatori logici, anch'essi coinvolti.

AND prende TOS e 2OS e li confronta bit per bit. Quando entrambi hanno un determinato bit nella condizione di verita', il bit corrispondente nel risultato e' messo in condizione di verita'. Altrimenti il bit e' messo in condizione di falsita'. Il risultato viene posto nello stack.

OR lavora in modo simile, ma mette in verita' il bit del risultato se TOS o 2OS hanno un bit vero in quella posizione.

XOR lavora anch'esso in modo simile ma mette un bit vero nel risultato se il bit corrispondente in TOS e 2OS e' diverso. Un uso interessante e' di ottenere un risultato con il bit segno messo a uno se i segni di TOS e 2OS sono diversi.

Alcuni dei derivati aritmetici sono piuttosto semplici. 1+ e' equivalente a 1 +, e aggiunge 1 a TOS. Analogamente 2+ e' equivalente a 2 + e aggiunge 2 a TOS.

Poi abbiamo +- che rimuove TOS e nega il nuovo TOS se il TOS originale era negativo. E' implementato chiamando MINUS se TOS era minore di zero. La versione per i numumeri doppi e' D+- , che rimuove TOS e nega il numero doppio in 2OS e 3OS se TOS era negativo.

Vengono poi MIN e MAX . Prelevano TOS e 2OS, ne scartano uno e mettono il rimanente in TOS. MIN rimette il numero che era minore, MAX il maggiore. In entrambi i casi 2DUP crea una copia di TOS e 2OS da usare come base di confronto,e poi scarta uno degli inserimenti originali.

Alla fine, al primo livello di derivazione abbiamo M/MOD , che lavora come U/MOD tranne che lascia un numero doppio come risultato anziche' un numero singolo. E' utile lavorando con numeri grandi. La parola e' costruita come segue:

	Stack	
	a b c	
>R	a b	c in TORS
0	a b 0	fa di b un numero doppio
R	a b c 0	richiama c
U/MOD	a d e	divide b per c, quoziente e resto d
R<	a d e c	richiama c, cancella TORS
SWAP	a d c e	
>R	a d c	e in TORS
U/MOD	f g	a/d diviso per c, quoziente g resto f
R>	f g e	richiama e, cancella TORS

Vengono eseguite due divisioni. La prima e' "scalata" di un fattore di 65536, poiche' d in realta' e' la parte superiore di un numero doppio. Il quoziente e il resto vengono scalati allo stesso modo, cosi' e' possibile "concatenare" a e d nella forma di un numero doppio valido che e' ancora diviso per c. I due quozienti g ed e possono quindi essere "concatenati" come numero doppio.

Al secondo livello di derivazione, abbiamo ABS e DABS. ABS e' implementato da DUP +- . TOS viene duplicato e la copia rimossa. Se e' negativa, il TOS originale viene negato. Qualsiasi cosa accada, TOS risulta positivo, rimanendo conservato il valore assoluto. DABS usa analogamente 2DUP D+- per eseguire lo stesso processo su un numero doppio.

Giungiamo ora a M/ e M* . M* genera un numero doppio prodotto di due numeri singoli con segno. E' basato su U* , con le aggiunte atte a preservare i segni.

	Stack	
	a b	
2DUP	a b a b	
XOR	a b c	c e' positivo se a e b hanno lo stesso segno

```
>R          a b          c in TORS
ABS         a b          b diventa assoluto
AWAP        b a
ABS         b a          a diventa assoluto
U*          d e          prodotto
R>          d e c        c da TORS. TORS cancellato
D+-         d e          nega d/e se c e' negativo
```

La funzione XOR viene usata per definire un flag[3] indicante il segno del risultato richiesto, che e' contenuto nel Return Stack mentre viene eseguito il calcolo.

M/ e' leggermente piu' complicato:

```
                Stack
                a b c
OVER            a b c b
>R              a b c        b in TORS
>R              a b          c in TORS
DABS            a b          il doppio numero a b diventa
                             positivo
R               a b c        c copiato da TORS...
ABS             a b c        e fatto diventare positivo
U/MOD           d e          a/b diviso per c, quoziente e
                             resto d
R>              d e c        c da TORS. TORS=b
R               d e c b      b copiato da TORS
XOR             d e f        f positivo se b e c hanno lo
                             stesso segno
+-              d e          nega e se f e' negativo
SWAP            e d
R>              e d b        b da TORS, TORS cancellato
+-              e d          nega d se b e' negativo
SWAP            d e          resto d quoziente e.
```

I numeri a/b e c, ma non le loro copie nel Return Stack, vengono resi positivi e U/MOD calcola il valore assoluto del resto e del quoziente. Il quoziente viene negato se il segno di b e c differisce. Al resto viene dato il segno di b.

Al prossimo livello abbiamo delle semplici derivate. */MOD combina la moltiplicazione e la divisione. Rimuove tre numeri dallo stack, tutti come numeri singoli. 2OS e 3OS vengono moltiplicati fra di loro, e il numero doppio prodotto viene diviso per TOS. I numeri vengono considerati con segno durante tutta l'elaborazione.

```
                Stack
                a b c
>R              a b          c in TORS
M*              d e          doppio numero prodotto
R>              d e c        c da TORS. TORS cancellato
M/              f g          f resto, g quoziente
```

/MOD rimuove TOS e 2OS. Li tratta come numeri singoli, divide 2OS per TOS e mette il resto, poi il quoziente, nello stack. Essenzialmente, e' M/ adattato a lavorare su un numero singolo:

3 **Letteralmente "bandiera", é una variabile (a volte un singolo bit) usata per indicare una condizione di varietá o di falsitá assumendo, in genere, i valori 0 o 1. (N.d.T.)**

	Stack a b	
>R	a	b in TORS
S->D	a c	il segno esteso a formare un numero doppio
R>	a c d	b ripreso da TORS. TORS cancellato
M/	d e	a/c diviso per b, resto d, quoziente e.

Ora, finalmente, arriviamo al moltiplicatore semplice * . E' implementato da M* DROP, la parte superiore del numero doppio viene scartata. Analogamente il divisore semplice / e' implementato da /MODE SWAP DROP che scarta il resto.

Per finire, arriviamo a */ e MOD . Il primo e' implementato da */MOD SWAP DROP , il resto viene scartato. Il secondo e' implementato da /MOD DROP , che rimuove il quoziente.

Questi operatori sono stati esaminati in dettaglio perche' le semplici definizioni, possono a volte sembrare ambigue e anche perche' le definizioni dettagliate mostrano come possono essere costruite le parole FORTH , a mo' di piramide, per eseguire azioni complesse.

Dato che le definizioni compilate forniscono indirizzi associati espliciti per ogni funzione che chiamano, la funzione richiesta puo' essere trovata molto velocemente e anche una complessa struttura a piramide di molti strati puo' essere eseguita rapidamente, quantunque solo le "primitive", in effetti, svolgano il lavoro necessario.

In questo paragrafo, abbiamo esaminato:

AND	Rimuove TOS,2OS. Mette nello staK un AND a 16 bit delle due parole.
OR	Come AND , ma e' una funzione OR.
XOR	Come AND , ma e' una funzione OR esclusivo.
1+	Aggiunge 1 a TOS.
2+	Aggiunge 2 a TOS.
+-	Rimuove TOS. Se e' negativo nega il nuovo TOS
D+-	Rimuove TOS, 2OS. Se il TOS originale e' negativo nega il nuovo TOS/2OS.
MIN	Scarta il maggiore fra TOS e 2OS.
MAX	Scarta il minore fra TOS e 2OS.
M/MOD	Rimuove TOS, 2OS, 3OS. Divide 2OS/3OS (considerati senza segno) per TOS. Mette il resto, quindi il quoziente nello stack come numero singolo con segno.
ABS	Da' a TOS un segno positivo, se necessario, per negazione.
DABS	Da a TOS/2OS un segno positivo, se necessario, per negazione.
M/	Rimuove TOS, 2OS, 3OS. Divide 2OS/3OS (considerati con segno) per TOS. Mette il resto nello stacK come numero singolo col segno del dividendo, quindi il quoziente con il segno appropriato a seconda dei segni di dividendo e divisore.

M* Rimuove TOS, 2OS. Mette il loro prodotto nello stack come numero singolo con segno.

*/MOD Rimuove TOS, 2OS, 3OS. Moltiplica 2OS per 3OS ottenendo come prodotto un numero doppio che divide per TOS. Tutti i numeri sono con segno. Mette il resto, quindi il quoziente, nello stack come numeri con segno.

/MOD Rimuove TOS, 2OS. Divide 2OS per TOS e mette il resto nello stack con il segno del dividendo, quindi il quoziente come numero singolo con segno.

* Rimuove TOS, 2OS. Li tratta come numeri singoli con segno, mette il loro prodotto nello stack come singolo numero con segno.

/ Rimuove TOS, 2OS. Divide 2OS per TOS e mette il quoziente nello stacK come numero con segno.

*/ Rimuove TOS, 2OS, 3OS. Moltiplica 2OS per 3OS e divide il numero doppio risultante per TOS. Mette il quoziente nello stack.

MOD Rimuove TOS, 2OS. Divide 2OS per TOS e mette il resto nello stack.

ACCESSO ALLA MEMORIA

L'accesso alla memoria dipende da sette primitive che permettono l'accesso a bytes, parole e parole doppie. Se queste vengono usate normalmente, la loro azione dettagliata non e' importante, ma se vengono usate in modi particolari, la loro implementazione puo' diventare significativa. Le primitive sono:

C! Rimuove TOS, 2OS. TOS definisce l'indirizzo della locazione di un byte. Il byte basso di 2OS viene posto in quella locazione.

! Rimuove TOS, 2OS. TOS definisce l'indirizzo della prima di due locazioni di byte consecutive. Il byte basso di 2OS viene messo nella prima locazione, il byte alto nella seconda. Queste locazioni non dipendono dalla macchina.

2! Rimuove TOS, 2OS, 3OS. TOS definisce l'indirizzo della prima di quattro locazioni di byte consecutive. Il byte basso di 2OS e' messo nella prima locazione, il byte superiore nella seconda. Il byte basso di 3OS e' messo nella terza locazione, il byte superiore nella quarta. Queste locazioni non dipendono dalla macchina.

C@ Rimuove TOS. TOS definisce l'indirizzo della locazione di un byte, il contenuto della quale e' messo nello stack. (Byte superiore a 0)

@ Rimuove TOS. TOS definisce l'indirizzo della prima di due locazioni di bytes consecutive. Viene formata una parola dal contenuto della prima locazione considerato come byte inferiore e

dal contenuto della seconda locazione considerato come byte superiore e tale parola viene messa nello stack.

2@ Rimuove TOS. TOS definisce l'indirizzo della prima di quattro successive locazioni di byte. Vengono messe nello stack due parole. Il byte inferiore della seconda parola viene dalla prima locazione, il byte superiore dalla seconda. Il byte inferiore della prima parola (che diventa 2OS) viene dalla terza locazione, il byte superiore dalla quarta. Queste locazioni non dipendono dalla macchina.

+! Rimuove TOS, 2OS. TOS definisce l'indirizzo della prima di due locazioni di byte. Conservando le convenzioni adottate per ! e @ , i contenuti di 2OS vengono sommati ai contenuti delle due locazioni.

? non e' una primitiva (= @ .) ma puo' essere convenientemente trattata qui. Essa scrive il contenuto di una locazione definita da TOS.

Queste parole possono essere usate con indirizzi espliciti, ma e' generalmente piu' conveniente usare un nome di variabile. Alcune variabili, come BASE, sono definite sin dall'inizio. Altre saranno menzionate quando le incontreremo.

Chiamando un nome di variabile, viene messo l'indirizzo associato nello stack, cosi' n BASE ! mettera' il valore n nella variabile BASE, mentre BASE @ mettera' il contenuto di BASE nello stack. le nuove variabili vengono create cosi':

```
Y VARIABLE nome
```

Questo definisce una locazione doppia a 16 bit, l'indirizzo della quale verra' messo nello stack in risposta al nome. Alla variabile e' assegnato il valore iniziale di Y. Per esempio:

```
10 VARIABLE PILA
```

Definira' una variabile con valore iniziale 10 (interpretato secondo il valore corrente di BASE) e la variabile puo' essere letta con PILA @ o ridefinita con PILA ! .

Per variabili di numeri doppi, il formato e':

```
YY VARIABLE nome
```

Vengono riservate quattro locazioni di byte e messe al numero doppio di YY.

Non ci sono condizioni standard per definire variabili di byte, ma puo' essere definita un'estesa area di memoria usando ALLOT . Quando viene definita una nuova parola nel "dizionario", l'inserimento necessario viene fatto riferendosi al puntatore DP. ALLOT muove in avanti il puntatore, estendendo lo spazio disponibile. Per esempio:

```
X VARIABLE nome 6 ALLOT
```

definira' una variabile a due byte, con l'arrangiamento necessario per creare l'accesso ad essa. Il puntatore del dizionario si muovera' quindi avanti di sei locazioni, che sono dunque riservate per l'estensione dei dati nella variabile. Le locazioni non vengono cancellate. Gli otto bytes cosi'

riservati potrebbero essere usati per un array di quattro parole cui si puo' accedere tramite:

```
n nome SWAP DUP + +
```

SWAP porta n a TOS, dove viene duplicato e il risultato viene sommato all'indirizzo definito da nome. Il risultato finale sara' l'indirizzo del byte inferiore dell'n-esimo elemento dell'array. Aggiungendo @ verra' letto il contenuto dell'elemento. Sarebbe anche possibile usare lo spazio riservato per contenere otto bytes, dove l'indirizzo dell'n-esimo byte sarebbe ottenuto da:

```
n nome +
```

Devono sempre essere usate le parole di lettura e scrittura corrette per le dimensioni di memoria scelte, questo e' un esempio di come la flessibilita' del FORTH ponga delle esigenze all'utente.

Possono essere anche definite delle costanti. Queste si comportano in modo piuttosto diverso dalle variabili, in quando chiamando una costante si pone nello stack il valore della costante, non il suo indirizzo. Siccome non c'e' il problema di assegnare valori alle costanti, chi ha bisogno di sapere dove si trovano? Il formato delle definizioni per costanti a numeri singoli e doppi sono:

```
X CONSTANT nome
XX 2CONSTANT nome
```

Uno degli usi di una costante e' definire un indirizzo al di fuori del FORTH. Il sistema operativo dello Spectrum definisce tre bytes chiamati FRAMES, che contengono il tempo (espresso in cinquantesimi di secondo) trascorso da quando il calcolatore e' stato acceso, con la possibilita' di essere messi a qualsiasi valore si desideri. I tre bytes possono essere letti in combinazione tramite:

```
23672 @ 23674 C@
```

che definisce un numero doppio.

Piuttosto che fare affidamento sulla memoria per gli indirizzi esatti, e' possibile definire il primo indirizzo in una costante:

```
23672 CONSTANT FRAMES
Le locazioni possono allora essere lette con:
FRAMES DUP @ SWAP 2+ C@
L'azione dello stack e':
FRAMES   23672
DUP      23672    23672
@        23672   (23672)  Contenuto della locazione 23672
SWAP    (23672)   23672
2+      (23672)   23674
C@      (23672)  (23674)  Contenuto della locazione 23672
```

Questo lascia un numero doppio nello stack che rappresenta il valore corrente della variabile FRAMES. Puo' essere visualizzato usando D.

Notate come gli indirizzi spariscano non appena vengano utilizzati, lasciando sgombro il risultato richiesto.

L'accesso al sistema operativo al di fuori del FORTH puo' fornite estensioni utili, ma e' necessaria prudenza. Nel caso della variabile dello Spectrum FRAMES, per esempio, possono essere letti valori non corretti. La variabile viene aggiornata ogni 20 mS, e dopo 20 minuti e 50,72 secondi i due bytes inferiori raggiungono 65535. Al prossimo incremento, essi tornano a zero e il terzo byte viene incrementato; se questo avviene fra il momento in cui vengono letti i bytes inferiori e il momento in cui viene letto il byte superiore, ne risultera' un grosso errore.

Questo puo' essere controllato chiamando ancora FRAMES2 immediatamente, il risultato non dovrebbe differire dal precedente di piu' di uno.

Comunque, mentre vi tenete a mente questo, potreste trovare il modo di usare M/MOD e U/MOD per visualizzare il tempo trascorso in ore minuti e secondi.

In aggiunta alle parole usate per accedere a particolari locazioni o gruppi di locazioni, ci sono parole usate per trattare aree piu' ampie.

FILL rimuove TOS, 2OS e 3OS. Iniziando dalla locazione definita da 3OS, 2OS bytes sono riempiti col codice definito nel byte inferiore di TOS. Per esempio, se un blocco di memoria e' stato definito con:

```
0 VARIABLE ARRAY n ALLOT
```

le locazioni nel blocco possono essere messe a zero da:

```
ARRAY n 2+ 0 FILL
```

ARRAY mette l'indirizzo di inizio in 3OS, 2OS e' messo a n+2, il numero di bytes (inclusi i due definiti da VARIABLE), e 0 definisce il codice da mettere in ogni locazione.

ERASE e' in effetti 0 FILL , consentendo a quanto sopra di diventare:

```
ARRAY n 2+ ERASE
```

Analogamente BLANKS e', in realta', 32 FILL e riempie l'area con il codice degli "spazi".

CMOVE considera TOS come un contatore, 2OS come un indirizzo destinazione e 3OS come un indirizzo sorgente e copia il numero di bytes definiti da TOS dall'area dalla destinazione in su nell'area dalla sorgente in su. Non deve essere usato se la sorgente e la destinazione si sovrappongono, con la sorgente sotto la destinazione, poiche' la sorgente sarebbe modificata prima di essere copiata.

TOGGLE puo' essere incluso in questo gruppo di parole. Rimuove TOS e 2OS, usando 2OS per definire la locazione di un byte. I contenuti della locazione subiscono un OR esclusivo (XOR) con il byte minore di TOS, cambiando lo stato di ogni bit che e' vero in TOS. Creato originariamente per servire a una determinata funzione, TOGGLE puo' essere utile in altri contesti.

Nello Spectrum, per esempio:

```
23697 2 TOGGLE
```

cambiera' il bit 1 di PFLAG e abilitera' o disabilitera' l'OVER.

Riassumendo le parole che sono state trattate:

C!	Rimuove TOS, 2OS. Mette il byte basso di 2OS a TOS
!	Rimuove TOS, 2OS. Mette 2OS a TOS
2!	Rimuove TOS, 2OS, 3OS. Mette 2OS/3OS a TOS
C@	Rimuove TOS. Lo sostituisce con il contenuto della locazione del byte TOS
@	Rimuove TOS. Lo sostituisce con il contenuto della locazione della parola TOS
2@	Rimuove TOS. Lo sostituisce con il contenuto della locazione della doppia parola TOS
?	Rimuove TOS. Scrive il contenuto della locazione della parola TOS, come numero singolo con segno.
X VARIABLE nome	Inizializza una variabile a due byte contenente X, con l'indirizzo della locazione associato al nome
XX 2VARIABLE nome	Inizializza una variabile a quattro bytes contenente XX, con l'indirizzo della locazione associato al nome
ALLOT	Rimuove TOS. Riserva i prossimi TOS bytes nel dizionario
X CONSTANT nome	Inizializza una costante con valore X associato al nome
XX 2CONSTANT nome	Inizializza una costante a due parole con valore XX associato al nome
FILL	Rimuove TOS, 2OS, 3OS. Mette il codice del byte inferiore di TOS in 2OS locazioni che iniziano da 3OS
ERASE	Rimuove TOS, 2OS. Mette zero in TOS locazioni che iniziano da 2OS
BLANKS	Rimuove TOS, 2OS. Mette il codice dello spazio in TOS locazioni che iniziano da 2OS
CMOVE	Rimuove TOS, 2OS, 3OS. Copia TOS bytes dall'area da 3OS in su nell'area da 2OS in su
TOGGLE	Rimuove TOS, 2OS. Modifica il valore contenuto nella locazione del byte 2OS facendo un XOR col byte inferiore di TOS.

FUNZIONI SPECIALI DELLO SPECTRUM

In generale, il FORTH e' indipendente dalla macchina su cui lavora. A parte variazioni nel vocabolario fondamentale, ci sono generalmente alcune caratteristiche speciali che dipendono dalla macchina. Le funzioni descritte qui sono particolari per il

FORTH Abersoft, che ha inoltre un vocabolario generale piu' vasto di altre implementazioni.

Per cominciare, ci sono alcune funzioni di controllo dello schermo e opzioni grafiche, che sono molto simili a quelle del BASIC Spectrum, tranne che i parametri precedono le istruzioni anziche' seguirle. Per funzioni relative alla posizione del carattere, per esempio, la linea e' definita da 2OS, la colonna da TOS.

Y X AT	stabilisce la posizione di stampa alla linea Y colonna X
Y X ATTR	mette nello stack il byte di attributi della linea Y colonna X
Y X SCREEN	mette nello stack il codice ASCII del carattere alla linea Y colonna X, tranne quando e' un carattere definibile dall'utente
	La definizione dei colori e' semplice:
X INK	dove X = 0 - 9
X PAPER	dove X = 0 - 9
X BORDER	dove X = 0 - 7

I numeri hanno lo stesso significato che in BASIC.

BRIGHT, FLASH, INVERSE e GOVER sono abilitati da un TOS non-zero e disabilitati da un TOS zero. Dunque 0 BRIGHT disabilita il BRIGHT mentre 1 BRIGHT lo abilita. GOVER e' l'OVER in BASIC cui e' stato modificato il nome per ovvie ragioni, avendo OVER in FORTH un significato proprio. Per la grafica ad alta risoluzione, 2OS specifica la coordinata X e TOS la coordinata Y. Per esempio, 20 100 PLOT equivale alla forma BASIC PLOT 20,100.

DRAW, tuttavia, lavora su base assoluta, anziche' relativa, una linea viene disegnata in una posizione specifica, comunque sia andata prima.

POINT ritorna 1 o 0 in TOS a seconda dello stato del punto specificato sullo schermo.

Una caratteristica particolare delle istruzioni della grafica a punti e' che non appaiono messaggi di errore se si eccedono i limiti dello schermo, ma il processo di plotting continua con valori di coordinate limitati, e se permettete una escursione eccessiva potreste dover aspettare parecchio prima che la traccia riappaia.

UDG mette nello stack l'indirizzo di inizio dell'area dei caratteri definibili dall'utente.

Per finire c'e'BLEEP, che e' sigificativamente differente da BEEP. Il TOS definisce il tono e 2OS la durata ma, mentre c'e' una maggiore flessibilita' di quello direttamente disponibile in BASIC, il sistema e' piu' difficile da usare. Sono necessari dei calcoli per ottenere la scala musicale, su questa base:

Per generare una frequenza di F Hz, il TOS deve essere calcolato come segue:

```
TOS=(437500/F)-30
```

Guardando nell'altra direzione:

```
F=437500/(TOS+30)
```

La durata della nota e' determinata dal numero di cicli, cosi' 2OS deve essere F x T , dove T e' la durata in secondi.

Va notato che se viene selezionata una frequenza molto bassa e una lunga durata, il sistema sembra rimanere sospeso, poiche' la routine BEEPER in BASIC funziona e funziona...; senza che l'utente possa usare BREAK.

BREAK (CAPS SHIFT e SPACE) funzionera' in alcune circostanze, ma e' prevista una alternativa da CAPS SHIFT e 1. Questo, tuttavia funzionera' solo se avete scritto il programma prevedendone l'uso. Se non l'avete fatto, potreste accorgervi che il vostro programma e' entrato in un loop infinito.

Un altro punto degno di menzione e' che puo' essere usata la ZX Printer col FORTH. Questa viene accesa con 1 LINK e spenta con 0 LINK.

RIASSUNTO DEL CAPITOLO I

Fino ad ora sono state definite piu' di ottanta parole, ma ne devono ancora essere menzionate altre duecento. Tuttavia, molte di queste ultime sono parole "di sistema", che non avrete bisogno di usare direttamente. Prima di proseguire e' bene acquisire famigliarita' con le parole gia' definite, provando varie combinazioni di esse e osservando i risultati.

Di tanto in tanto, potreste vedere un messaggio d'errore. Se tentate di leggere lo stack quando questo e' vuoto, avete l'errore 1. Se usate una parola che non e' nel dizionario, ottenete l'errore 0. Dopo ogni errore il sistema si cancella e diventa pronto per un nuovo inizio. Non dovreste incappare in altri errori a questo punto, a meno che non facciate qualcosa di piuttosto improbabile.

Vi accorgerete che non c'e' bisogno di inserire tutte le istruzioni insieme. I dati che inserite vanno nel buffer terminale di input. Quando premete ENTER o quando il buffer e' pieno, la funzione INTERPRET e' chiamata a interpretare cio' che avete detto. Per prima cosa provera' a confrontare un gruppo di caratteri delimitati da spazi con il nome di una parola del dizionario, e se cio' fallisce provera' a interpretare il gruppo come numero che sia valido rispetto alla base corrente. Se anche questo fallisce, visualizzera' l'errore 0.

INTERPRET lavora su una parola per volta, cosi', se premete ENTER dopo ogni parola o numero, il risultato sara' lo stesso. A volte e' preferibile una sequenza intera.

Se volete cancellare lo schermo prima di una determinata azione, dovete mettere un CLS nella stringa di parole e numeri che chiamano quella azione. Non dimenticate che il sistema insiste nell'aggiungere un "ok" in coda a ogni output che completa una stringa di azioni. Potete mettere 0 0 AT vicino al termine della stringa per mettere l'"ok" nell'angolo in alto a destra dello schermo.

Se volete sapere quello che sta' accadendo nello stack senza perdere alcun dato, un trucco utile e' usare:

```
ROT DUP . ROT DUP . ROT DUP .
```

Cio' visualizzera' 3OS, 2OS, e TOS nell'ordine, ma lascera' lo stack inalterato. Dovreste essere in grado di capire perche'.

Che cosa possiamo fare con le parole che abbiamo esaminato? Ben poco, potreste dire. Potremmo effettuare qualche funzione aritmetica complessa, ma tutto in forma intera, e cio' puo' sembrare limitante. Con un po' di pazienza scoprirete che questa non e' necessariamente una limitazione seria. Un'occhiata alla routine DRAW listata nell'Appendice A vi dara' un cenno delle possibilita' in tale direzione. I cambiamenti in X e Y sono calcolati in unita' di 0,0000152, usando il principio della graduazione, che verra' esaminato in dettaglio in seguito.

In termini di esplorazione delle possibilita' del FORTH, abbiamo appena iniziato. Come preludio a ulteriori progressi, dobbiamo vedere il modo di funzionamento del dizionario, e il modo in cui possiamo cominciare a creare parole per conto nostro.

CAPITOLO II:

IL DIZIONARIO

Questo capitolo descrive la struttura del dizionario e il modo in cui il FORTH usa il dizionario per rispondere alle istruzioni. Coloro che sono impazienti di procedere oltre possono saltalo per il momento, ma questo capitolo aiutera' a spiegare il meccanismo coinvolto nelle funzioni che saranno descritte.

IL DIZIONARIO

Il caricamento del nastro FORTH predispone un'area di memoria col Dizionario FORTH, che contiene tutte le parole essenziali al sistema. Durante l'inizializzazione vengono definite alcune variabili fuori dall'area del dizionario e vengono stabiliti gli stacks e i buffers, ma il dizionario rimane il cuore del sistema.

Il nome "dizionario" e' preciso e appropriato, poiche' si riferisce a una lista di parole e alle loro definizioni. Quando inserite una parola FORTH, il sistema la ricerca nel dizionario ed esegue l'azione definita. Con almeno 260 parole da controllare, questo impiega del tempo, il che e' accettabile tenendo conto che il processo di inserimento della parola ha richiesto piu' tempo. E' necessario un processo piu' rapido per eseguire un programma.

Le definizioni di alcune parole sono cosi' complesse che costituiscono da sole dei piccoli programmi. La parola M/ chiama quindici altre parole e alcune di queste ne chiamano di altre. Per permettere che cio' sia eseguito velocemente, le definizioni sono messe nella forma di indirizzi associati alle definizioni delle altre parole, cosicche' una semplice routine in codice macchina puo' saltare direttamente all'area richiesta. Poiche' non c'e' bisogno di cercare parole oltre alla prima, il FORTH non perde tempo nel muoversi da un processo all'altro. Ecco perche' e' cosi' veloce.

Per far si che un tale sistema lavori efficentemente, gli inserimenti nel dizionario devono essere fatti con cura. Ogni parola e'costituita da otto "campi" (fields).

Per primo c'e' il Campo Nome (Name Field). Questo comincia con un byte detto byte lunghezza che - fra le altre cose - da' il numero di lettere del nome che e' stato definito. Siccome la lunghezza massima permessa e' 31 lettere, questa puo' essere definita dai bit 0 - 4 del byte lunghezza.

Il bit 5 del byte lunghezza e' il bit "macchia" (smudge). Quando questo e' vero, la parola non sara' riconosciuta per valida anche se sara' listata come presente, se viene chiamato VLIST per vedere il contenuto del dizionario. Il bit viene messo a uno durante la creazione di una nuova parola, e messo a zero

quando la definizione e' completa, cosi' che definizioni incomplete vengono marcate come non valide.

Il bit 6 del byte lunghezza e' vero quando indica "precedenza". Cio' significa che la parola sara' sempre eseguita, anche durante la creazione e l'inserimento di una nuova parola nel dizionario, quando il sistema e' detto essere nel modo "compilazione" e considera tutte le altre parole che trova come facenti parte della nuova definizione.

Il bit 7 del byte lunghezza e' sempre vero.

La parola stessa, in codice ASCII, completa il Campo Nome, con il bit 7 del codice dell'ultima lettera messo a uno per indicare la fine della parola.

Quando viene eseguita una ricerca nel dizionario, deve essere esaminata a turno ogni definizione. Per semplificare l'operazione, il campo nome e' seguito immediatamente dal Campo Associazione (Link Field) che contiene l'indirizzo della prossima definizione da esaminare. La ricerca comincia con la parola inserita piu' di recente, che si trova in cima al dizionario, e procede in giu' per la memoria, controllando un campo nome e prelevando l'associazione se non viene riscontrata corrispondenza. Questo utilizza una variabile chiamata HERE, che e' definita dal puntatore principale del dizionario.

Il processo di associazione e' illustrato in Fig. 1, dalla quale si puo' vedere che il numero di locazioni che devono essere scandite e' mantenuto a un minimo assoluto.

Quando viene trovata la parola richiesta, entra in gioco il terzo campo. Questo e' il Campo Codice (Code Field) e contiene un'associazione al codice macchina che deve essere eseguito per implementare la funzione. Il codice in se' puo' essere ovunque nel dizionario ma, quando copre l'esecuzione completa della funzione, segue generalmente il Campo Codice. In alcuni circoli, le parole che vengono eseguite interamente in linguaggio macchina sono dette "primitive", poiche' sono gli elementi di base che eseguono il lavoro effettivo, essendo chiamate nella sequenza richiesta da parole di livello piu' alto.

Molte parole, tuttavia, sono definite solamente tramite riferimenti ad altre parole e, per esse, il campo codice e' associato a una routine che interpreta il quarto campo, il Compo Parametri (Parameter Field).

Il campo parametri contiene una serie di associazioni ai Campi Codice di altre parole, alcune associazioni sono seguite dai dati necessari all'azione delle altre parole. Per esempio, c'e' una parola, LIT, che e' un'abbreviazione di LITERAL. La sua funzione e' quella di mettere il dato seguente del Campo Parametri allo stack. Analogamente, la parola ." e' seguita nel campo parametri dal testo che deve essere visualizzato. In questo modo, il campo parametri, costituisce una completa definizione del programma per le parole che serve.

Alcune delle parole FORTH piu' misteriose, esistono principalmente per aiutare ad interpretare il dizionario. TRAVERSE, per esempio, muove il puntatore di scansione (conservato temporaneamente in 2OS) dalla fine di un campo nome ad un altra. Prendendo TOS e 2OS dallo stack, somma TOS a 2OS

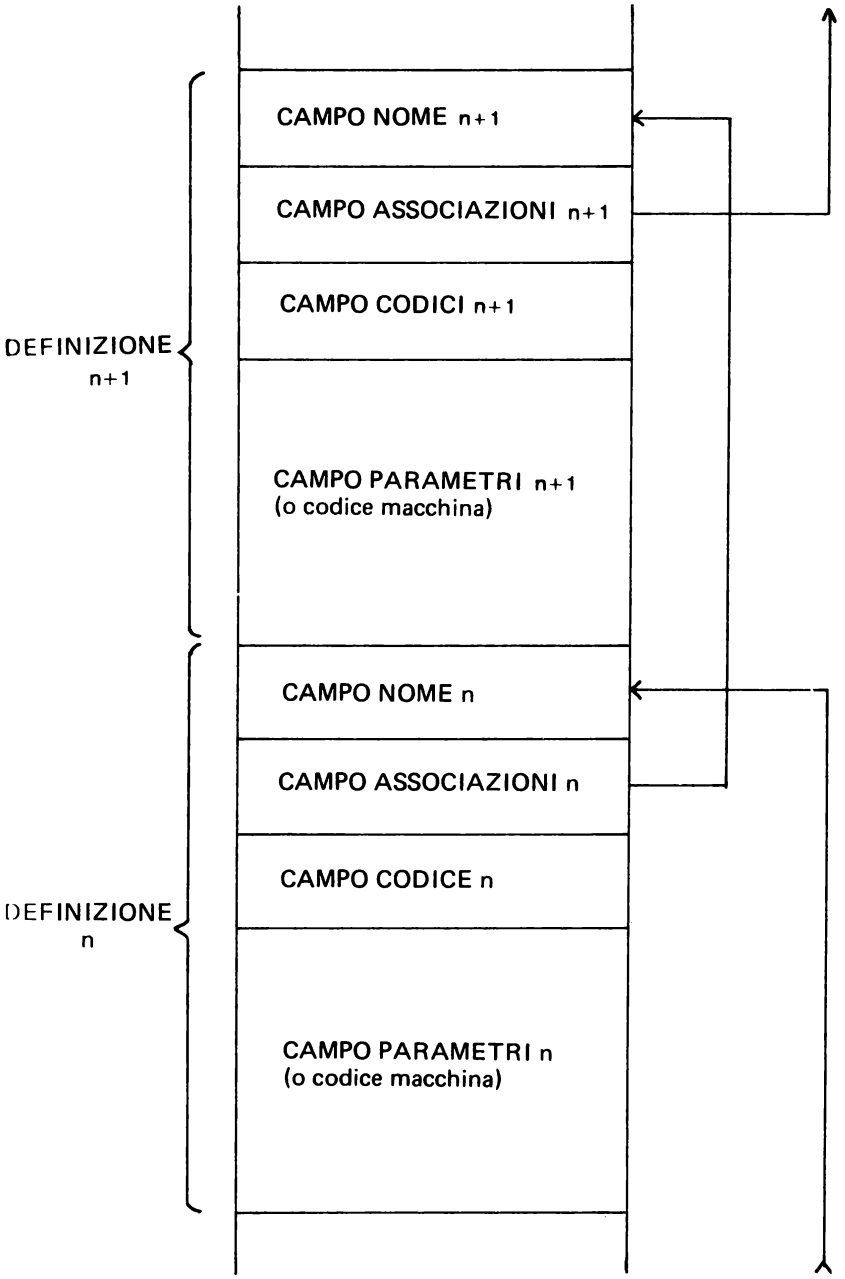


FIG. 1: FORMATO DEL DIZIONARIO MOSTRANTE IL SISTEMA DI SCANSIONE

ripetitivamente finche' non viene trovato un byte col bit 7 vero. Se TOS e' 1, TRAVERSE scandisce in avanti, se e' -1, la scansione e' verso il basso. Il valore risultante del puntatore e' lasciato in TOS.

TRAVERSE viene generalmente chiamato come parte di un processo. La parola NFA , per esempio, converte il TOS alll'indirizzo di un campo parametri all'indirizzo del campo nome associato e ha la forma:

```
5 - - 1 TRAVERSE
```

Sottraendo cinque dall'indirizzo del campo parametri porta il TOS indietro, passando il campo codice e il campo associazione, ognuno dei quali contiene due bytes, fino alla fine del campo nome. La combinazione -1 TRAVERSE, quindi, esegue una scansione all'indietro fino all'inizio del campo nome.

PFA esegue la conversione inversa, la sua forma e':

```
1 TRAVERSE 5 +
```

Il campo nome e' scandito fino alla fine, e quindi l'aggiunta di cinque porta il TOS all'indirizzo del campo parametri.

Aggiungere una parola al dizionario e' semplice. Se inserite:

```
23672 CONSTANT FRAMES
: FRAME FRAMES DUP @ SWAP 2+ C@ ;
```

scoprirete che avete creato una nuova costante chiamata FRAMES e una nuova parola, FRAME, che puo' essere usata per mettere il contenuto delle variabili dello Spectrum, FRAMES, nello stack, come parola doppia.

I due punti, chiamano il modo compilazione, cosi' invece di eseguire le parole che seguono, il sistema inserisce una nuova parola nel dizionario col nome FRAME, definendo il campo parametri in modo che chiami DUP @ SWAP 2+ C@ in sequenza (vedere capitolo III per maggiori dettagli sulle definizioni tramite due punti).

E' ora possibile leggere la varabile chiamando la sola parola FRAME , invece di chiamare le singole parole di seguito. Se volete, e' possibile usare la parola FRAME in un'ulteriore definizione, giacche' questa si comporta esattamente allo stesso modo del resto delle parole del dizionario.

Il punto e virgola termina il processo, facendo ritornare il sistema al modo esecuzione.

Per trovare le corrette associazioni per il campo parametri delle nuove parole, e' necessario ricercare attraverso il dizionario le parole relative, cosi' la compilazione e' relativamente lenta, ma l'esecuzione della parola compilata sara' veloce. Una volta definita una parola, potete usarla per definirne altre, e troverete che ci sono diverse parole che possono essere usato solo nelle "definizioni tramite due punti".

Un punto importante, concerne il bit "macchia" del byte lunghezza nel campo nome. Quando inizia una definizione, il bit e' a uno per indicare una definizione non valida, e non e' rimesso a zero finche' la definizione non e' completata. Se qualcosa va male, come scoprire una parola inesistente nella definizione, il bit e' lasciato a uno.

La parola puo' allora essere trovata da VLIST , che lista le

parole nel dizionario, ma non viene riconosciuta.

Incidentalmente, il bit "macchia" viene manipolato da SMUDGE, che e' definito come LATEST 32 TOGGLE. LATEST lascia l'indirizzo del campo nome creato piu' di recente, e 32 (=20H) seleziona il bit 5 come quello che deve essere invertito.

Avendo definito FRAME, potreste volerla usare in altre definizioni:

```
: MINSEC 50 M/MOD ROT DROP 60 U/MOD 0 60 U/MOD . . . ;
: CLOCK FRAME MINSEC ;
```

Cominciando dalla seconda definizione, questa chiama FRAMES quindi MINSEC .

FRAME mette i contenuti della variabile FRAMES nello stack come numero doppio, e MINSEC prima divide il numero per 50 per dare il numero di secondi dall'accensione. Il resto indesiderato e' scartato da ROT DROP e quindi 60 U/MOD separa i minuti e i secondi. Poiche' questo produce un numero singolo come risultato, viene aggiunto 0 allo stack per generare un numero doppio e un ulteriore 60 U/MOD genera le ore e i minuti come numeri separati. Le tre cifre ottenute vengono visualizzate una dopo l'altra.

Chiamando la singola parola CLOCK verra' dunque visualizzato il tempo trascorso da quando il calcolatore e' stato acceso, in ore, minuti, e secondi. Se vi sentite ambiziosi potete tentare di definire una parola che inizializzi la variabile FRAMES nell'area BASIC per avere il tempo reale.

Se questo esame del formato del dizionario ha solleticato la vostra cueriosita', potreste dare una scorsa al sommario del dizionario standard in Appendice A, che da' le definizioni di tutte le parole standard. Scoprirete che una variabile viene inserita allo stesso modo di una parola esecutiva, con una routine speciale per interpretare il campo parametri, mentre una costante e' anch'essa simile nel formato ma anch'essa usa una routine speciale.

Poiche' una variabile occupa almeno otto bytes dello spazio del dizionario, e' comprensibile che l'approccio ideale sia di ridurre al minimo l'uso delle variabili. Tuttavia, con tanto spazio disponibile e' permesso essere un poco stravaganti.

Questa escursione nel meccanismo del FORTH non era un fattore essenziale per l'uso del linguaggio, ma dovrebbe aver dato un'utile visione di cio' che avviene all'interno del sistema. Quando saranno state comprese tutte le parole disponibili, si vedra' che la precedente descrizione e' stata molto semplificata. Invece di riferirsi a parole esistenti per formare altre parole, e' possibile inserire routines in codice macchina per svolgere determinati compiti. Il processo di compilazione puo' essere interrotto per permettere al sistema di calcolare dati per ulteriori compilazioni, quindi la compilazione puo' essere ristabilita.

Per ottenere il massimo dal sistema e' meglio limitare l'ampiezza di ogni definizione creata piuttosto che cercare di stipare tutto in un'unica definizione. Cosi' facendo, la validita' di ogni nuova parola puo' essere controllata prima di procedere ulteriormente e, oltre tutto, puo' essere evitata

confusione.

Menzionare la possibilita' di controllare le nuove parole ci porta naturalmente alla domanda su come possono venir corretti gli errori. Cio' verra' trattato in dettaglio in un capitolo successivo, per il momento e' sufficente dire che e' sempre possibile "dimenticare" (FORGET) una parola definita. Se si tratta della parola definita piu' di recente, questa sola sara' cancellata dal dizionario semplicemente mettendo i puntatori di inizio ricerca al campo nome della definizione precedente. Se in seguito sono state definite altre parole, tutte queste andranno perdute. In entrambi i casi, tuttavia, il dato rilevante rimane inalterato. Questo e' semplicemente ignorato dal processo di ricerca. Cio' puo' essere paragonato all'uso di OLD dopo NEW, in alcune versioni del BASIC, per riprendere un programma distrutto accidentalmente.

Ci sono molti modi di usare il FORTH. Per cominciare, potete basarvi sulla costruzione di una piramide di definizioni del dizionario, senza usare trucchi fantasiosi. Aumentando la confidenza potrete essere indotti a esplorare tecniche piu' complesse, e potrete eventualmente cominciare a vedere come possono essere modificate le parole inserite nel dizionario per far fare loro cio' che volete. Se va vostra ambizione vi conduce su questo sentiero, procedete con cautela piuttosto che velocemente poiche' ci sono delle trappole per gli incauti, ma verra' fatto un tentativo per fornirvi tutte le informazioni di cui avrete bisogno.

In questo capitolo abbiamo incontrato:

1 TRAVERSE	Converte il TOS dall'indirizzo di inizio del campo nome all'indirizzo di fine del campo nome.
-1 TRAVERSE	Converte il TOS dall'indirizzo di fine del campo nome all'indirizzo di inizio del campo nome.
CFA	Converte il TOS dall'indirizzo del campo parametri all'indirizzo del campo codice.
LFA	Converte il TOS dall'indirizzo del campo parametri all'indirizzo del campo concatenazione.
PFA	Converte il TOS dall'indirizzo del campo nome all'indirizzo del campo parametri.
NFA	Converte il TOS dall'indirizzo del campo parametri all'indirizzo del campo home.
LATEST	Mette il contenuto di CURRENT nello stack (Identificando l'indirizzo del campo nome dell'ultima parola inserita nel dizionario).
SMUDGE	Cambia lo stato della locazione definita da LATEST eseguendo un XOR del TOS (byte inferiore) con il contenuto di quella locazione.
TOGGLE	Cambia il contenuto della locazione definita dall'indirizzo in 2OS facendo l'XOR col byte basso di TOS.
FORGET	Cambia il contenuto di CURRENT per puntare al campo nome della parola del dizionario identificata dal campo concatenazione della parola che segue FORGET.

CAPITOLO III

Definizioni tramite due punti

Questo capitolo spiega come possono essere create nuove parole usando le "definizioni tramite due punti", e introduce parole inerenti i processi di comparazione (branching) e ricorsivita' (looping). Se l'uso del FORTH fosse limitato solo all'esecuzione di parole inserite via tastiera, sarebbe davvero un linguaggio molto limitato, ma l'intero concetto si basano sulla possibilita' di introdurre nuove parole nel dizionario principale o in un dizionario sussidiario creato per scopi particolari. Una volta che sia stato fatto un nuovo inserimento, questo puo' essere utilizzato per aiutare a definire ulteriori parole, finche' un intero programma puo' essere definito in termini di una sola parola.

Nella compilazione di nuove definizioni, ci sono un certo numero di parole che sono inaccettabili per l'esecuzione diretta, e queste saranno esaminate a tempo debito. Prima, tuttavia, alcuni esempi dell'uso di definizioni create aiuteranno a mostrare le possibilita' del sistema.

ESTENDERE IL DIZIONARIO

Nel paragrafo sull'accesso alla memoria, dicevamo che l'inserimento di una variabile poteva essere esteso dall'uso di ALLOT per riservare un'area che potrebbe contenere un'array. L'accesso ad un determinato elemento dell'array richiede una corta routine che trova l'indirizzo dell'elemento:

```
n nome SWAP DUP + +
```

Siccome n, il numero dell'elemento richiesto e il nome, il nome dell'array sono fattori variabili possiamo definire utilmente:

```
: ACALC SWAP DUP + + ;
```

Possiamo ora ottenere l'indirizzo richiesto inserendo il piu' breve:

```
n nome ACALC
```

Questo avra' lo stesso effetto della sequenza definita prima. Possiamo ora fare un passo avanti e definire:

```
: A! ACALC ! ;
: A@ ACALC @ ;
```

Avendo definito ACALC possiamo ora usarlo per definire due parole che scrivono e leggono dall'indirizzo dell'elemento dell'array. Cio' condensa ulteriormente il codice richiesto. Sarebbe stato possibile definire A! e A@ direttamente, senza prima definire ACALC, ma qui stiamo trattando i principi piuttosto che la pratica, e l'itinerario seguito dimostra il principio molto chiaramente.

In un programma trattato piu' avanti, richiederemo un array a tre elementi chiamato PILE che sara' definito da:

```
0 VARIABLE PILE 4 ALLOT
```

Possiamo quindi accedere all'elemento n dell'array con:

```
n PILE A@
```

o scrivere in esso con:

```
n PILE A!
```

Dobbiamo stare attenti a non porre n maggiore di 2, altrimenti usciremmo dall'area riservata. Gli indici dell'array, dovrebbe essere stato notato, vanno da 0 a 2 e non da 1 a 3 come nel BASIC Spectrum. Se chiamassimo 3 PILE A! scriveremmo nel campo nome della successiva parola del dizionario.

Gli array multidimensionali sono piu' difficili da maneggiare. La regola usuale per calcolare un elemento di numero n dati gli indici e le dimensioni e' basato sulla applicazione iterativa di:

$n_x = n_{x-1} * dimensione_x + indice_x$

Inizialmente, $n_0 = 0$, cosi' $n_1 = indice$

Quindi $n_2 = indice_1 * dimensione_2 + indice_1$

Questo processo e' ripetuto a turno per ogni indice.

Per un array a tre elementi, la sequenza FORTH sarebbe:

```
I3 I2 I1 D2 * + D3 * + ;
```

Cio' metterebbe nello stack il numero dell'elemento. Potremmo quindi definire:

```
: ELEMENT D2 * + D3 * + ;
```

Potremmo ora leggere dagli elementi definiti dagli indici S1, S2, S3 chiamando:

```
S3 S2 S1 ELEMENT ARRAY A@
```

Dove ARRAY e' il nome dato all'array.

In questi esempi piuttosto semplici, noterete che ogni definizione inizia con due punti, per ordinare al sistema di creare una nuova parola da inserire nel dizionario nel modo compilazione e termina con punto e virgola per ordinare il ritorno al modo esecuzione.

La parola che segue i due punti e' considerata il nome della nuova parola e le parole che seguono sono poste nel campo parametri in termini dei loro indirizzi al campo codice. Notate che due punti e punto e virgola non sono necessari quando si definiscono variabili e costanti.

E' necessario che paventiate la possibilita' di riempire lo spazio disponibile creando troppe definizioni. Potete sempre usare FREE per controllare quanto spazio vi rimane, e vi accorgerete che questo si restringe piuttosto lentamente. Se volete calcolare quanti bytes occupa una nuova definizione, sommate:

- Il numero di lettere del nome della nuova parola piu' 1.
- Due bytes per il campo associazione.
- Due bytes per il campo codice.
- Due bytes per ogni parola usata, incluso il punto e virgola ma esclusi i due punti.
- Quattro bytes per ogni valore numerico.
- Quattro bytes per una comparazione o ricorsione (vedi paragrafo seguente).

La parola ELEMENT , definita piu' sopra, richiede 32 bytes.

Dovreste ora essere in grado di poter sperimentare la costruzione di alcuni semplici gruppi di parole. Quando create una nuova parola, annotate come questa modifichi lo stack. ACALC rimuove n e un indirizzo, lasciando al loro posto un indirizzo. A! rimuove n, l'indirizzo e un numero singolo, mentre A@ rimuove n e l'indirizzo e lascia un nuovo numero singolo. ELEMENT rimuove tre indici e lascia n.

Se annotate questi cambiamenti, vi accorgerete che e' facile seguire tutte le modifiche dello stack, visto che non c'e' bisogno di passare tutte le definizioni parola per parola per vedere quello che accade.

Per il momento, sarete ostacolati dal fatto che le vostre definizioni, subendo uno scroll, escono rapidamente dallo schermo e vengono perse, ma un rimedio a cio' verra' spiegato nel capitolo IV, dove viene descritto il sistema del disco RAM, che e' un sistema di immagazzinamento nel quale potrete salvare le vostre definizioni e riprenderle ogniqualvolta vi sorga un dubbio o vogliate correggerle.

DIRAMAZIONI E RICORSIVITA'

Esponenti della programmazione strutturata, considerano meritoritevole il fatto che il linguagio FORTH non offra una funzione paragonabile alla parola BASIC GOTO, che considerano insana. Infatti, sarebbe molto difficile introdurre direttamente una tale parola, visto che non ci sarebbe modo di definire la destinazione. All'interno del dizionario, tuttavia, la maggior parte delle funzioni di diramazione (branching) e di ricorsione (looping) sono implementate tramite salti relativi, che sono parenti stretti di GOTO.

Le funzioni di salto sono:

BRANCH XXXX ,che trasferisce l'azione ad un indirizzo associato aggiungendo XXXX al puntatore interpretativo.

0BRANCH XXXX , che agisce allo stesso modo, ma solo se TOS e' zero, altrimenti non ha effetto.

Siccome XXXX e' una parola a 16 bit, sarebbe teoricamente possibile altare ad ogni altro punto del dizionario, ma il raggio d'azione dei salti effettivamente usati, e' relativamente piccolo.

BRANCH , da solo, e' di uso limitato, ed e' generalmente usato per definire un'azione alternativa in associazione a 0BRANCH. Per poter apprezzare l'azione di 0BRANCH , dobbiamo considerare alcuni dei modi particolari in cui deve essere definito il TOS per determinare se 0BRANCH agisce o no.

Per prima cosa ci sono i calcoli aritmetici semplici. Se mettono a zero il TOS quando viene chiamato 0BRANCH , questo agira': altrimenti non fara' niente, in entabi i casi il TOS verra' rimosso.

Poi vengono gli operatori logici, che non sono molto diversi:

=	Mette una condizione non-zero in TOS se TOS=2OS, altrimenti lo mette a zero.
<	Mette una condizione non-zero in TOS se 2OS e' minore di TOS, altrimenti zero.
>	Mette una condizione non zero nel TOS se 2OS e' maggiore di TOS, altrimenti zero.
U<	Come < , ma considerando senza segno i numeri.

Ognuno di questi operatori rimuove TOS e 2OS, e viene perso quando agisce 0BRANCH . E' uso riferirsi alla condizione messa in TOS dall'operatore come ad un flag' vero (non-zero) o ad un flag falso (zero).

Poi ci sono gli operatori che agiscono su un solo valore piuttosto che basarsi su un confronto fra TOS e 2OS:

0<	Mette una condizione non-zero in TOS se il TOS e' negativo, altrimenti zero.
0=	Mette una condizione non-zero in TOS se il TOS e' zero, altrimenti zero.

Entrambi questi operatori rimuovono il TOS originale.

L'operatore 0= in effetti inverte lo stato del flag in TOS e viene anche chiamato NOT . Puo' essere usato dopo altri operatori per invertire il loro senso. Un certo numero di altre funzioni lasciano flags. Per esempio ?TERMINAL mette una condizione non-zero nello stack se viene chiamata da tastiera la combinazione BREAK (CAPS SHIFT e SPACE o CAPS SHIFT E 1). Di tanto in tanto incontreremo altri esempi.

DIRAMAZIONI CONDIZIONATE

Un tipo di diramazione familiare e ingannevole consiste in:

IF...ELSE...ENDIF

ma la sua azione puo' apparire meno familiare. Essa ricorre nella forma:

condizione IF azione1 ELSE azione2 ENDIF azione3.

Se la condizione (in TOS) e' vera, vengono eseguite le azioni 1 e 3, se la condizione e' falsa, vengono eseguite le azioni 2 e 3. Il flag condizione e' rimosso dallo stack, ed e' a volte conveniente crearlo con -DUP che duplica il TOS solo se non e' zero. Il TOS originale sarebbe quindi disponibile per l'azione 1 ma sarebbe scartato se fosse eseguita l'azione 2.

L'implementazione nel dizionario dovrebbe essere lineare. IF e' sostitutuito da 0BRANCH che esegue un salto all'azione 2 se la condizione e' zero. Altrimenti, un BRANCH al termine dell'istruzione 1 esegue un salto all'azione 3.

La compilazione di IF...ELSE...ENDIF viene confrontata con un numero di riferimento che e' controllato da ?PAIR . IF definisce il numero 2 e ELSE richiede 2 per essere definito. Anche ELSE definisce il numero 2 e ENDIF richiede 2 per essere definito. E' anche possibile omettere ELSE e l'azione 2 se e' richiesta solo l'azione condizionale.

Se vi sembra piu' espressivo, potete sostituire ENDIF con THEN; hanno esattamente lo stesso significato.

Il ciclo "DO" del FORTH e' simile al ciclo "FOR" del BASIC, essendo la forma equivalente:

```
FOR N = A TO B ... NEXT          = B + 1 A DO ... LOOP
FOR N = A TO B STEP C ... NEXT   = B + 1 A DO ... C + LOOP
```

Notate che il parametro limite e' B+1 e non B. Mentre l'indice di iterazione e' minore del limite, LOOP rimanda l'azione ad un punto immediatamente successivo a DO , ma quando l'indice e' uguale al limite, vengono eseguite le parole che seguono LOOP. Il Return Stack contiene i dati necessari.

DO viene compilato come (DO). Quando viene eseguito (DO), il TORS viene trasferito a TORS (il valore iniziale dell'indice A) e 2OS a 2ORS (il valore limite B+1).

LOOP viene compilato come (LOOP). Quando (LOOP) viene eseguito, il TORS viene incrementato e il risultato viene comparato con 2ORS per vedere se il valore limite e' stato raggiunto. Se non e' cosi', la routine salta ad un punto immediatamente successivo a (DO) , l'ampiezza del salto e' calcolata nel momento in cui viene compilata la sequeza e posto nella parola successiva a (LOOP) nel campo parametri.

+LOOP viene compilato come (+LOOP) ,che agisce allo stesso modo di (LOOP) tranne che C (memorizzato nel campo parametri come una "letterale", un numero da mettere nel TOS) viene sommato all'indice invece dell'unita'.

All'interno di un ciclo DO, l'indice di iterazione e' normalmente in TOS ,a meno che non sia stato coperto da >R , nel qual caso il dato che lo copre deve essere rimosso da R> prima che venga raggiunto (LOOP). Anche la parola R mettera' TORS in TOS, senza disturbare il contatore. Per alcune inspiegabili ragioni, viene comunemente usato un sinonimo di R : I . Le due parole agiscono esattamente allo stesso modo, usando anche lo stesso codice macchina. Usando I , il valore corrente dell'indice puo' essere usato in calcoli all'interno del ciclo.

Il valore di 2ORS puo' essere copiato in TOS da I' ,dando il valore limite dell'indice; questi possono tornare utili quando il valore richiesto e' (limite-indice). A volte e' necessario usare il contatore d'iterazione di un ciclo DO piu' esterno, quando si abbiano due cicli nidiati . Per cio', e' richiesto 3ORS, che puo' essere copiato in TOS dalla parola J .

Riguardo a cio', c'e' un sottile tranello. Se il valore del Return Stack viene letto all'interno di una definizione chiamata in un ciclo DO , viene aggiunta una associazione di ritorno al Return Stack durante l'esecuzione della definizione ed e' necessario scavare un po' piu' a fondo, usando I' anziche' I e J invece di I'. Per esempio:

```
    : CALC2 I' . ;
    : CALC1 10 0 DO CALC2 LOOP ;
Ora CALC1 stampera' i numeri da 0 a 9.
Cosi', il listato di CALC1 dopo la definizione sara':
    : CALC1 10 0 DO I . LOOP ;
```

L'associazione di ritorno mostra dove deve essere ripresa l'esecuzione nella sequenza di chiamate nella tabella dei paramtri.

In BASIC, e' permesso a volte uscire da un ciclo FOR NEXT, a volte invece e' imprudente. Per uscire da un ciclo DO prematuramente, viene usata la parola LEAVE . Questa parola mette l'indice uguale al valore del limite, terminando il ciclo al successivo LOOP. LEAVE viene comunemente usato nel formato:

condizione IF LEAVE ENDIF

Meno familiari agli utenti delle vecchie versioni del BASIC, ma implementate in alcune delle versioni piu' recenti, sono le restanti forme di cicli:

BEGIN azione UNTIL

ripete l'azione finche' questa non mette la condizione di verita' nello stack. Un modo comune per permettere l'uscita manuale dal ciclo e':

BEGIN azione ?TERMINAL UNTIL

Premendo CAPS SHIFT e 1, ?TERMINAL mettera' una condizione di verita' in TOS, cosi' il ciclo verra' interrotto.

Se non ci sono condizioni prima di UNTIL , il ciclo puo' continuare per sempre.

Notate che UNTILL rimuove il TOS essendo implementato da 0BRANCH che riporta l'azione a subito dopo BEGIN se trova che il TOS e' 0. Potete usare END al posto di UNTIL se lo desiderate; hanno lo stesso significato. Questo e' un altro esempio di sinonimi in FORTH, parole differenti che significano la stessa cosa e vengono eseguite dallo stesso codice macchina.

Una struttura piu' complessa e' fornita da :

BEGIN azione1 WHILE azione2 REPEAT azione3

WHILE rimuove il TOS. Se TOS=0 , azione1 e' seguita da azione3. Atrimenti vengono eseguite azione1 e azione2 alternate.

Il formato BEGIN azione AGAIN deve essere usato con cautela poiche' non v'e' modo di uscire dal ciclo se non chiamando QUIT, ABORT o EXIT, eseguite in modo condizionale con l'azione. Avete usato QUIT fin dall'inizio poiche' questa e' la routine che accetta inputs dall'utente. Essa cancella il Return Stack mentre ABORT cancella entrambi gli stacks, ed e' piu' drastico. EXIT rimuove il TORS, distruggendo l'associazione che sarebbe altrimenti usata per dirigere ulteriori azioni. Quello che succede poi dipende da cosa emerge come nuovo TORS. All'interno di un ciclo LOOP, potrebbe trattarsi del contatore del ciclo, e cio' causerebbe mutilazioni. Sperimentate con EXIT cautamente, anche se nella peggiore delle ipotesi, dovrete ricaricare il nastro FORTH e iniziare di nuovo...

Per finire, c'e' la struttura CASE , un'aggiunta speciale al fig-FORTH che viene omessa in alcune implementazioni. Il formato e':

```
condizione   CASE
             A OF azione1 ENDOF
             B OF azione2 ENDOF
             C OF azione3 ENDOF
             ecc.
             ENDCASE
```

Se A corrisponde alla condizione data, viene eseguita l'azione 1. Se B corrisponde alla condizione data, viene eseguita l'azione 2, e cosi' via. La condizione deve essere nella forma di un numero singolo (o un suo equivalente, ad esempio un codice ASCII). A, B e C possono essere parole o sequenza di parole che producono un risultato simile. Ne vedremo un esempio verso la fine del programma dato nell'ultimo capitolo di questo libro.

STRINGHE E SIMILI

Fino ad ora, abbiamo parlato soltanto di numeri, dicendo poco o nulla dei testi. E' ora di rimediare a questa deficenza.

La parola ." compila una parola nel dizionario che contiene il testo che segue. Per esempio:

```
: ABC ." Programma numero 1 " ;
```

definisce una parola nel dizionario che scrive il testo quando viene chiamata la parola ABC. Notate che ci deve essere uno spazio fra ." e l'inizio del testo e che il testo e' completato da ulteriori "virgolette".

La stessa definizione puo' includere numeri e altre parole, permettendo definizioni del tipo:

```
: BCD 10 6 AT ." Programma No 2 " ;
```

che posizionera' il testo sullo schermo alla linea 10, colonna 6.

Notate che la parola ." viene sostituita dall'associazione alla parola compilata (.") e segue un carattere di conteggio. L'output effettivo e' eseguito da TYPE ,che visualizza TOS caratteri presi da un area di memoria con inizio a 2OS (TOS e 2OS vengono rimossi). Il carattere di conteggio e' definito da WORD che che stabilisce il numero di caratteri e che inserisce il numero, con il testo, nel dizionario. In genere non c'e' bisogno di preoccuparsi di queste funzioni insite nel sistema.

Il FORTH non ha , come caratteristica di base, alcuna possibilita' per la manipolazione delle stringhe. Se volete qualcosa del genere, dovete costruirvelo per conto vostro. La procedura dovrebbe essere in grado di individuare la stringa sulla quale volete operare, identificare la parte della stringa che vi interessa ed estrarre questa parte, o per l'uso immediato, o per formare una nuova parola. Un esempio illustrera' meglio il metodo.

Definiamo una stringa di base:

```
: DATE ." GenFebMarAprMagGiuLugAgoSetOttNovDic " ;
```

La sequenza -FIND DATE mettera' il TOS a 1 ad indicare che la parola DATE e' stata trovata, 2OS conterra' la lunghezza in bytes della parola e 3OS conterra' l'indirizzo del campo parametri della parola. Tutto cio'che ci serve sono gli indirizzi:

```
-FIND DATE DROP DROP 3 + CONSTANT POINTER
```

Questo definira' una costante che punta all'indirizzo piu' 3, che e' l'inizio del testo memorizzato. Abbiamo saltato i due

bytes che definiscono ." e il byte lunghezza. Notate che questa e' una sequenza da eseguirsi immediatamente, non e' una definizione tramite due punti.

Per prelevare l'n-esimo mese, dobbiamo calcolare 3*(n-1) e aggiungerlo alla costante POINTER ottenendo il primo carattere del nome del mese. L'indirizzo e' messo nella variabile OUT :

```
: DATEADD 1 - 3 * POINTER + OUT ! ;
```

Per ottenere l'inizio dell'n-esimo mese, dobbiamo fare n DATEADD , cosi' possiamo costruire:

```
: OUTDATE DATEADD 3 0 DO OUT @ @ EMIT LOOP ;
```

I contenuti di OUT vengono incrememtati da EMIT , che emette i caratteri messi in TOS da OUT @ @ . Il nome intero del mese puo' essere ora scritto da n OUTDATE .

La proceduta descritta piu' sopra puo' sembrare piuttosto complicata all'inizio, ma ha il vantaggio di essere estremamente flessibile.

Il FORTH prevede anche alcune possibilita' di formattamento dei numeri in scrittura . Le parole coinvolte sono:

<#	Mette in HLD l'indirizzo del buffer per l'output del testo, come contenuto in PAD. Tale buffer non ha un indirizzo fisso ma fluttua 68 bytes sopra la cima del dizionario.
#	Opera su un numero doppio in TOS/2OS, generando la cifra meno significativa della sua rappresentazione nella base numerica corrente, e lasciando il residuo in TOS/2OS come numero doppio. La cifra e' memorizzata in PAD, viene usato HLD come puntatore in decremento.
#S	Ripete # finche' il numero doppio viene ridotto a zero.
SIGN	Usato fra <# e #> , inserisce un segno meno davanti ad una stringa numerica convertita se 3OS e' negativo. 3OS viene scartato ma TOS/2OS rimangono indistrubati. Essi conterranno normalmente il numero doppio sul quale si e' operato.
#>	Completa la formazione di una stringa numerica scartando il numero doppio e lasciando gli indirizzi della sua locazione e un byte lunghezza. Generalmente e' seguito da TYPE.
TYPE	Emette TOS caratteri presi dalla memoria, partendo da 2OS.

Queste parole permettono di stabilire un formato rigido. Dal momento che #S terminera' l'output quando il doppio numero raggiunge zero, puo' essere definito un modello usando # che inserira' gli zero superflui. C'e' un'altra parola:

HOLD Decrementa HLD e memorizza TOS nella stringa numerica come carattere in codice ASCII. Per esempio, 46 HOLD inserira' un punto decimale.

Sperimentando, potrete capire meglio le piene possibilita' di queste parole. Notate che viene usato un puntatore in decremento per porre le stringhe nel buffer, dal momento che TYPE usa un puntatore in incremento. La cifra meno significativa viene

generata per prima ma mostrata per ultima.

Il processo puo' usare un ciclo DO per chiamare # DPL volte prima che venga inserito il punto decimale, DPL contiene la posizione del punto decimale. SIGN viene per ultimo cosi' che il risultato viene visualizzato o stampato prima.

Conviene menzionare qui alcune delle altre parole relative alle stringhe e all'output dei testi.

COUNT , usato con il TOS che punta al byte lunghezza di un testo in memoria, mette l'effettivo byte lunghezza in TOS e l'indirizzo d'inizio del testo che segue in 2OS. Puo' quindi essere usato TYPE per emettere il testo.

.CPU visualizza il nome del computer, nel caso vi siate dimenticati quale macchina state usando ...

-TRAILING modifica il byte lunghezza della stringa di un testo numerico per rimuovere gli spazi superflui. Richiede il byte lunghezza originale in TOS e l'indirizzo dell'inizio del testo in 2OS. Lascia la stessa situazione con il byte lunghezza aggiustato. Puo' essere dunque interposto fra COUNT e TYPE , o puo' precedere TYPE quando e' usato per emettere una stringa formattata.

.ID considera il TOS come l'indirizzo di un campo nome ed emette il nome della relativa definizione.

Le funzioni puramente dedicate alle stringhe in FORTH non sono troppo estese, essendo questo visto principalmente come linguaggio per il calcolo, ma puo' essere fatto perecchio con le opzioni fornite, pensandoci un po'.

INPUT/OUTPUT

L'input di dati per l'esecuzione immediata o per le definizioni tramite due punti e' cosi' semplice che tende ad essere dato per scontato, ma viene fatto parecchio lavoro, immediatamente, per ottenere tale semplicita'.

Viene usata la routine QUIT per tali inputs, poiche' agisce ogniqualvolta vengono completate altre azioni e deve essere restituito il controllo all'utente. Questa chiama QUERY che mette nello stack l'indirizzo del buffer terminale di input, e quindi il numero 80H che e'l'ampiezza del buffer in bytes. Viene quindi chiamato EXPECT per trasferire i caratteri dalla tastiera al buffer di input terminale finche' viene premuto Return o sono stati inseriti ottanta caratteri. Quindi QUERY azzera il puntatore IN , cosi' che' i dati che sono stati inseriti possano essere scanditi.

Questi dati sono nella forma di codici ASCII, e devono essere interpretati. Ogni gruppo di caratteri che inizia e termina con uno spazio puo' essere una parola FORTH, e viene eseguita una ricerca nel dizionario per controllare questa possibilita'. Se la ricerca e' negativa, il gruppo di caratteri puo' essere un numero; viene chiamata NUMBER per controllare. NUMBER richiede l'indirizzo di un puntatore in TOS e controlla che sia un numero valido a seconda del valore corrente di BASE, cercando anche il punto decimale. Se quest'ultimo viene trovato, DPL contiene il

numero di cifre che seguono, altrimenti contiene -1. Se il numero non e' valido si ha un errore.

Se DPL e' positivo, INTERRUPT mette un numero doppio nello stack, altrimenti mette un numero singolo. Se il gruppo di codici non viene identificato ne' come parola ne' come numero, INTERPRET riporta un errore.

Un punto interessante e' che la parola potenziale viene inserita nel dizionario sia che sia valida o meno, ma il puntatore del dizionario non viene avanzato finche' la parola non e' stata controllata, cosi' le parole senza senso non hanno un'esistenza reale e possono essere in seguito sovrascritte.

Se viene riconosciuta una parola nel modo diretto, questa viene eseguita, mentre i numeri vengono messi nello stack, viene cosi' mantenuta una continuita' d'azione, anche se l'inserimento del testo avviene in piccole sezioni. Nel modo compilazione, la compilazione avviene in modo simile.

Una situazione piuttosto diversa si ha durante l'esecuzione di parole compilate. La tastiera viene ignorata a meno che non venga specificamente richiesto un input. Cio' puo' essere fatto tramite:

KEY	Mette il codice ASCII del tasto premuto in TOS.
INKEY	Come KEY ma se non viene premuto alcun tasto mette FFH in TOS.

La differenza e' che KEY aspetta che si prema un tasto, mentre INKEY no.

Conviene usare queste parole per controlli semplici, per rispondere S o N, o per interrompere l'azione finche' non siete pronti. E' possibile tradurre il codice ASCII in un valore numerico usando DIGIT , che richiede BASE in TOS e il carattere in 20S:

```
KEY BASE @ DIGIT
```

Questo mettera' nello stack il valore numerico, quindi un flag vero se il codice e' un numero valido oppure mettera' nello stack un flag falso se il codice non e' valido. Comunque il numero non verra' visualizzato se non espandete la sequenza cosi:

```
KEY DUP EMIT BASE @ DIGIT
```

Sarebbe possibile estendere ulteriormente la sequenza in modo che accetti e combini un certo numero di cifre per formare un numero completo, ma la e' piu' conveniente usare la combinazione QUERY INTERPRET.

L'output dei dati sul display e' stato adeguatamente trattato altrove, e l'uso di 1 LINK per accendere la stampante e di 0 LINK per spegnerla e' gia' stato menzionato. Le funzioni input/output che restano riguardano le porte:

INP	Rimuove il TOS e lo considera il numero della porta, e mette il TOS al valore del byte letto dalla porta.
OUTP	Rimuove il TOS e lo considera il numero della porta, rimuove 20S e lo considera come dato. Il dato viene emesso dalla porta specificata.

E' necessaria una certa prudenza nel selezionare le porte,

poiche' alcune di queste hanno un uso dedicato nel sistema operativo dello Spectrum. I bits da 0 a 4 dell'indirizzo della porta dovrebbero essere alti, e il bit 7 basso, il che lascia solo quattro indirizzi utilizzabili: 1FH, 3FH, 5FH e 7FH. Le limitazioni si riferiscono particolarmente alle porte in output.

RIMANENZE

Sono gia' state menzionate un gran numero di parole importanti. Quelle associate al processo di compilazione verranno trattate in seguito, nel capitolo IV, ma verra' fatto qui un tentativo per collegare fra loro le parole che restano.

Per prima cosa c'e' la parola FORTH in se'. Questa chiama il vocabolario FORTH, distinguendolo da ogni altro set di parole che possa essere stato definito, tipo vocabolario EDITOR. Chiamando il nome di un vocabolario, vengono attivate le associazioni all'interno del dizionario, diventa cosi' disponibile il set di parole' richiesto. Per essere precisi, viene rimesso a zero CONTEXT, portando il puntatore al punto d'inizio per le ricerche. Un tentativo di usare FORGET per eliminare una parola che non e' nel vocabolario corrente, da' errore 24, perche' potreste distruggere molte piu' parole di quante vorreste. In ogni caso, non vi sara' permesso di usare FORGET quelle parole che si trovano aldila' del punto definito da FENCE che agisce come una bariera protettiva.

Quando usate FORGET su una parola, non scartate solo la parola in causa ma anche tutte le parole che vengono dopo nel dizionario, cioe' tutte le parole di origine piu' recente. Cio' viene talvolta usato vantaggiosamente per compilare la parola posticcia TASK come prima parola di un programma. FORGET TASK scartera' quindi il programma, comprese tutte le parole valide che contiene. Tali parole non possono essere scartate direttamente.

La variabile STATE determina il modo corrente di lavoro, esecuzione diretta o compilazione e genera un errore se viene usata una parola non appropriata a tale stato. Il sistema di controllo degli errori usa un certo numero di parole, un buon punto di partenza e' ?ERROR, che ricorre nella forma di:

f n ERROR, dove n definisce il numero dell'errore ed f e' il flag che e' vero se l'errore e' presente. Se il flag e' falso, f ed n vengono cancellati dallo stack e l'azione continua.

Se il flag e' vero, viene chiamato ERROR con n in TOS. Quello che succede poi dipende dal contenuto di WARNING. Se WARNING e' negativo viene chiamato (ABORT), essendo ABORT chiamato a turno. Altrimenti viene visualizzata la parola sbagliata seguita da un punto interrogativo, e quindi viene chiamato MESSAGE ancora con TOS=n, poi il puntatore dello stack viene reinizializzato, segue QUIT.

WARNING controlla anche l'azione di MESSAGE. Se WARNING=0, viene visualizzata la forma "MSG # n". Se WARNING e' maggiore di zero, il ristema visualizzera' un messaggio esplicito, essendo

il testo contenuto nel sistema di memorizzazione RAM-disc descritto nel capitolo IV. Siccome tale testo occuperebbe un quinto della memoria disponibile, non viene normalmente usato nel FORTH Abersoft, e non sarebbe affatto pratico per altre implementazioni che forniscano meno memoria.

?STACK controlla il valore del principale puntatore dello stack, leggendone il contenuto con SP@ e confrontando il risultato con S0, il suo valore inizializzato, e con HERE + 80H. Se il puntatore e' sotto HERE + 80H c'e' il rischio di un conflitto con il programma in memoria e viene chiamato ?ERROR con n=7 e un flag vero. Se il puntatore e' sotto S0, lo stack e' stato svuotato e viene chiamato ?ERROR con n=1 e un flag vero. Se nessuno di queste condizioni esiste, viene posto un flag falso e, anche se n=7, ERROR non ha effetto. L'approccio a una condizione di errore 7 puo' essere anticipato usando FREE . per vedere la quantita' di memoria libera.

Altri controlli di errore sono:

?COMP chiama l'errore 17 se il sistema non e' nel modo compilazione.

?EXEC chiama l'errore 18 se il sistema non e' nel modo di esecuzione diretta.

?PAIRS chiama l'errore 19 se le parole di ramificazione e di ricorsione non sono accoppiate nella sequenza corretta.

?CSP chiama l'errore 20 se il puntatore dello stack non corrisponde al contenuto della variabile CSP, indicando l'inserimento di una parola incompleta nel dizionario.

?LOADING chiama l'errore 22 se il buffer di input terminale e' in uso.

Le funzioni sussidiarie e le variabili associate al sistema di errori sono:

!CSP	Mette CSP al valore corrente del puntatore dello stack.
R0	La sorgente di inizializzazione per il puntatore del Return Stack.
RP@	Legge il puntatore del Return Stack e lo mette in TOS.
RP!	Mette il puntatore del Return Stack al valore di R0.
SP!	Mette a 20 il puntatore dello stak del calcolatore.

Per finire, in questo gruppo, puo' venire chiamato WHERE dopo un errore durante la compilazione, in seguito a cio' verra' localizzata la sorgente dell'errore evidenziando la linea relativa sul display con una freccia che punta alla parola sbagliata. Il verdetto non e' sempre accurato. Se, per esempio, vengono dimenticati i due punti all'inizio di una definizione tramite due punti,l'errore puo' evidenziarsi solo quando viene raggiunto il punto e virgola. Tenendo questo a mente, WHERE vi fornisce un utile aiuto per il debugging[4] del codice sorgente.

Poche delle parole appena descritte, verranno normalmente chiamate dall'utente. Come molte altre, vengono fornite per essere usate dal sistema interno FORTH.

COLD e WARM, sono un'altra faccenda. Queste permettono di

4 Operazione che consiste nell'eliminare gli errori di vario genere che, inevitabilmente, incorrono nella stesura del programma. (N.d.T.)

reinizializzare il sistema con e senza la perdita delle estensioni del dizionario. Possono essere chiamate come parole, ma sono anche accessibili dal BASIC con GOTO 2 per COLD e GOTO 3 per WARM. Il sistema BASIC puo' essere inserito con MON.

E' a volte utile sapere quanto sia grande il dizionario, visto che puo' essere salvato su nastro in forma estesa, e SIZE serve a questo scopo, mettendo in TOS il numero di bytes fra ORIGIN e HERE. ORIGIN e' il punto di inizio nominale del programma e n+ORIGIN mette nel TOS l'indirizzo dell'n-esimo byte da quel punto iniziale.

EXECUTE e COMPILE producono l'inserimento del modo di lavoro nominato, mettendo STATE al valore appropriato.

NOOP non fa niente, tranne magari riempire un vuoto.

Ci rimane da dire che 0, 1, 2 e 3 sono parole FORTH sotto forma di costanti, il che evita la necessita' di calcolare il loro valore, e n USER fornisce l'indirizzo di una locazione all'interno dell'area variabili dell'utente, che e' definita in Appendice A.

L'appendice rispondera' anche a molte domande sulle parole che non sono ancora state trattate, includendo il vocabolario EDITOR , che contiene 28 parole. Tuttavia, gli esempi che troverete nel capitolo IV dovrebbero chiarire eventuali punti poco chiari.

CAPITOLO IV

Il Disco RAM

Questo capitolo tratta delle possibilita' di memorizzare il codice sorgente, per salvare e caricare copie da nastro, e descrive il vocabolario EDITOR.

GLI SCHERMI

Abbiamo bisogno ancora di un'altra caratteristica per rendere il FORTH pienamente vitale. L'inserimento di definizioni tramite due punti per la compilazione immediata, puo' essere estremamente tedioso, specialmente se sono necessarie frequenti modifiche. La soluzione si trova nello schermo, che memorizza il codice sorgente per la compilazione in maniera tale da consentire di fare cambiamenti in modo facile e piuttosto veloce.

Il FORTH Spectrum fornisce dodici schermi, ognuno contenente sedici linee di 64 caratteri. Questo e' il formato standard dello schermo FORTH, e non e' pienamente compatibile con il display, piu' piccolo, dello Spectrum. Cio' fa si che l'utente debba tollerare alcuni scroll e usare CAPS e 1 per il BREAK, ma il tutto e' piuttosto maneggevole.

L'input n LIST visualizzera' e selezionera' lo schermo n, ma il display iniziale visualizzera' linee piene di punti di domanda, poiche' l'area di memoria e' piena di bytes 0. INIT-DISC riempira' l'area di dello schermo del codice dello spazio. Quindi puo' essere inserito il codice sorgente, usando il vocabolario EDITOR, e puo' essere poi compilato da n LOAD dove n e' il numero dello schermo in causa. Se viene inserito --> alla fine dello schermo, la compilazione continuera' nel prossimo schermo. In questo modo, possono essere compilati tutti e dieci gli schermi (1-10), se lo si desidera.

SAVET salvera' su nastro l'intero contenuto del disco RAM, che puo' essere ricaricato con LOADT o controllato con VERIFY; tutte queste parole sono varianti di (TAPE), l'istruzione comune del sistema del nastro. Tutti i files su nastro hanno il nome "DISC", spettera' a voi trovare il modo di distinguerli l'uno dall'altro.

Cio' che puo' non essere immediatamente ovvio e' che possono essere caricati a turno un certo numero di gruppi di schermi, ognuno dei quali puo' essere compilato mentre e' presente, cosi' possono essere creati programmi relativamente grandi.

Le convenzioni sul FORTH, alcune rafforzate dalle caratteristiche del linguaggio, impongono alcune limitazioni sul modo in cui viene usato lo schermo.

Lo schermo 0 e' riservato a commenti e spiegazioni, e non puo' essere usato per la compilazione. Analogamente la linea 0 e' riservata al titolo, e x y INDEX scrivera' i titoli degli

schermi da x a y. Come per tutti i commenti, i titoli devono essere racchiusi fra parentesi tonde, con uno spazio dopo l'aperta parentesi, poiche' e' una parola FOTRH che significa "ignora tutto cio' che segue finche' trovi la parentesi chiusa". Le parentesi possono essere usate come inserimenti temporanei per forzare il sistema ad iniziare la compilazione in mezzo allo schermo, l'aperta parentesi viene messa all'inizio della linea 0 e la chiusa parentesi immediatamente prima del punto in cui la compilazione deve iniziare. La parola ;S causera' l'interruzione della compilazione, come un inserimento nullo in qualsiasi punto dello schermo.

Altre convenzioni derivano dal fatto che gli schermi furono originariamente intesi per lavorare come parte di un sistema a dischi, ed era ragionevole destinare due schermi al supporto dei testi dei messaggi d'errore. Gli errori da 1 a 15 erano forniti dallo schermo 4, i rimanenti dallo schermo 5. Questa caratteristica esiste ancora e puo' essere richiamata mettendo WARNING a 1 (1 WARNING !). Dovrete inserire i messaggi negli schermi interessati, ma questa puo' essere una cosa utile mentre vi abituate ai codici d'errore. Perderete due schermi, ma cio' puo' non essere troppo catastrofico all'inizio.

Per stampare i records, e' utile TRIAD. Esso visualizzera' o stampera' il conenuto di tre schermi, iniziando a 0, 3, 6, o 9. Cosi' 5 TRIAD stamperebbe il gruppo che include lo schermo 5, cioe' gli schermi 3, 4 e 5.

Ci sono un certo numero di parole che vengono fornite per essere usate nella manipolazione dello schermo, ma prima di esaminarle sara' meglio vedere il vocabolario dell'editor, che e' essenziale per definire i dati dello schermo.

L'EDITOR

Immediatamente dopo il caricamento del nastro FORTH, VLIST visualizzera' una stringa di parole iniziando da UDG, l'ultima parola inserita nel vocabolario FORTH di base. Definendo qualsiasi altra parola in questo vocabolario, questa apparira' prima di UDG essendo di origine piu' recente. Inserendo EDITOR e VLIST verra' mostrato un altro gruppo di parole all'inizio del display; queste sono le parole nel vocabolario speciale EDITOR. La parola EDITOR ha chiamato le associazioni nel dizionario per mettere in azione queste parole. La parola FORTH chiamera' altre associazioni cosicche' le parole dell'EDITOR non saranno piu' accessibili.

Ci sono forse troppe parole nel vocabolario EDITOR, ed e' meglio abituarcisi gradualmente. Per prima cosa dovete selezionare uno "schermo", cancellarlo, e metterlo in funzione. Se volete selezionare uno schermo senza cancellarlo, dovete usare n LIST. Indi, in qualunque momento, dopo che sia stato abilitato EDITOR, anche L visualizzera' lo schermo dalla linea 0.

Il metodo piu' semplice per inserire un testo in uno schermo vuoto e' il formato m P testo, questo mette il testo alla linea m dello schermo corrente.

Se poteste garantire il 100% di accuratezza nella battuta e una totale infallibilita' nella formazione delle definizioni tramite due punti, quest'unica istruzione sarebbe sufficente per la definizione del codice sorgente ma, inevitabilmente, ci saranno dei cambiamenti da fare. Il primo passo e' individuare il punto in cui va fatto il cambiamento e, per questo, esistono alcune caratteristiche utili alla ricerca, che dipendono dalla posizione del cursore, che e' memorizzata nella variabile R#.

TOP azzera le variabili, e dovrebbe essere usato prima di una ricerca, a meno di non essere sicuri che il cursore sia sopra o a sinistra del punto da cambiare. Supponete di voler cambiare FIRTH in FORTH. Se la parola sbagliata ricorre solo una volta nello schermo, TOP X FIRTH la cancellera', lasciando il cursore alla posizione che la parola occupava. C FORTH inserira' quindi la parola FORTH nello stesso punto. Ricordate che C e X, essendo parole FORTH richiedono che segua uno spazio prima dell'inizio del testo associato. Se inserite C senza il testo, inserirete un codice nullo che deve essere rimosso, poiche' agisce come un indicatore di fine. Fortunatamente, TOP X non solo localizzera' il codice nullo, ma lo sostituira' con uno spazio.

Se la parola che volete cancellare ricorre piu' di una volta, potete individuare la prima volta che ricorre, senza cancellarla, usando TOP F testo, e quini N andra' avanti alla prossima ricorrenza . Quando avrete trovato la posizione giusta, il cursore sara' alla fine del testo, ma B lo portera' ancora all'inizio del testo, cosi' che possano essere usate X e C.

TILL cancellera' tutti i testi dalla posizione del cursore alla fine della linea del cursore.

Il cursore puo' essere posizionato precisamente da n M , che muove il cursore di n posizioni e visualizza la posizione risultante. Il valore di n puo' essere positivo o negativo.

In tutti questi movimenti e cambiamenti, il testo operativo e' contenuto in un buffer chiamato PAD, e questo puo' essere usato per contenere temporaneamente le linee per poter modificare l'ordine delle altre linee dello schermo. Per poter muovere la linea n dello schermo corrente in PAD, avete bisogno di n H , mentre n D muove il testo e cancella la sorgente. Per contro, n E cancellera' le linee senza salvarle.

Quando una linea si trovi in PAD, e volete inserirla fra due linee adiacenti, n S muovera', di una linea verso il basso, la linea n e tutte le linee sotto di essa, perdendo la linea 15, e n R trasferira' il testo da PAD alla linea lasciata libera. Piu' semplicemente, n I eseguira' entrambe le operazioni.

Nel caso che lo schermo sia molto pieno, e' utile a volte impiegare n T , che batte una singola linea e la tiene in PAD; la linea puo' poi venir controllata separatamente.

Le restanti parole dell'editor, sono per la maggior parte sottofunzioni di quelle che abbiamo descritto, e non vengono generalmente usate direttamente, ma un'eccezione e' costituita da COPY.

La forma a b COPY , copiera' il contenuto dello schermo a nello schermo b. Una sua estensione usa un numero di schermo negativo, che copiera' uno schermo in o dallo spazio libero fra

il dizionario e lo stack. Se lo spazio libero e' limitato, e' meglio fare un tentativo d'secuzione per assicurarsi che l'area usata non sia occupata, facendo un trasferimento falso da essa.

Questa caratteristica ha il vantaggio che uno schermo puo' venir mosso da una registrazione su nastro ad un'altra. La registrazione che contiene originariamente lo schermo, viene caricata, e quindi lo schermo viene trasferito in uno spazio libero e cancellato dallo schermo sorgente. Il risultato viene salvato e verificato. Viene caricata quindi la seconda registrazione, e lo schermo viene ritrasferito in uno schermo libero. Essendo fuori dall'area di schermo normale, questo non ha subito l'effetto dal salvataggio e dal caricamento. Puo' essere ora salvato come parte della seconda registrazione.

Questo piccolo trucco puo' essere molto utile, ma funziona meglio quando non ci sono estensioni del dizionario, e lo spazio libero e' alla sua massima estensione.

Forse dovrebbe essere menzionato n DELETE . Questi cancella n caratteri alla sinistra della posizione del cursore, e viene in realta' fornito per servire X . Le parole che restano sono: MATCH , #LOCATE , #LEAD , #FLAG , -MOVE , -TEXT , 1LINE e FIND . Tutte queste sono principalmente parole per uso interno, piuttosto che parole per l'utente.

Se c'e' una pecca nel sistema di editing, sta nel gran numero di parole fornite. Con un po' di pratica, scoprerete che un piccolo sottogruppo di queste, servira' alla maggior parte degli scopi.

Sara' stato notato che le parole R e I del vocabolario EDITOR sono identiche a quelle del vocabolario FORTH. Per l'esecuzione diretta, tuttavia, le forme EDITOR, vengono trovate prima quando e' abilitato il vocabolario EDITOR, cosi' vengono usate queste.

Le parole per l'utente nel vocabolario EDITOR possono essere cosi' riassunte:

B	Muove indietro il cursore della lunghezza del testo contenuta in PAD.
C	Inserisce il testo seguente alla posizione del cursore, propagando il testo originale per fare spazio.
D	Rimuove TOS considerandolo come numero di linea e cancella quella linea dopo averla copiata in PAD.
DELETE	Rimuove TOS considerandolo come numero di caratteri e cancella quel numero di caratteri alla sinistra del cursore.
E	Rimuove TOS considerandolo come numero di linea e cancella quella linea con i codici dello spazio.
F	Ricerca il testo seguente dal cursore alla fine dello schermo.
FIND	Come F , ma usando un testo gia' in PAD, e termina con TOP.
H	Rimuove TOS considerandolo un numero di linea e copia quella linea in PAD.

I	Esegue S e R , inserendo la linea da PAD alla linea TOS.
L	Lista lo schermo corrente.
M	Aggiunge TOS alla posizione del cursore e visualizza la linea.
N	Trova il prossimo punto in cui ricorre il testo in PAD.
P	Mette il testo seguente alla linea definita da TOS.
R	Sostituisce la linea identificata da TOS con il testo in PAD.
S	Sposta di una linea verso il basso la linea identificata da TOS e le linee seguenti.
T	Visualizza la linea e la copia in PAD.
TILL	Cancella dal cursore alla fine della linea.
X	Cancella il testo seguente la prima volta che ricorre.
TOP	Mette a zero il cursore.
CLEAR	Cancella lo schermo identificato da TOS.

DIETRO GLI SCHERMI

Per conservare una compatibilita' almeno nominale con il fig-FORTH standard, il FORTH Spectrum include nel suo vocabolario un certo numero di parole che si riferiscono alle vere operazioni coi dischi, ma non sono direttamente relative al sistema del disco RAM.

Viene stabilita un'area per un buffer fra CBE0 e CFFF, e questa contiene otto aree di buffer di capacita' nominale di 128 bytes, altri quattro bytes vengono forniti per funzioni di controllo. In un vero sistema a dischi, questi sarebbero in diretta comunicazione con il disco, contenendo i records in lettura e fornendo i dati in scrittura.

In generale, le parole che si riferiscono a questi buffers possono essere ignorate tranne forse per fare esperimenti avventurosi. Le parole, tutte definite in dettaglio in Appendice A, sono:

Costanti:

# BUFF	Numero di buffers allocati (8)
B/BUFF	Numero di bytes per buffer (128)
B/SCR	Numero di blocchi per schermo (8)
C/L	Numero di caratteri per linea (64)
FIRST	Indirizzo d'inizio del buffer inferiore (CBE0)
HI	Indirizzo della fine dell'area schermo (FBFF)
LIMIT	Indirizzo della fine dell'area del buffer piu' 1 (D000)
LO	Indirizzo d'inizio dell'area dello schermo (D000)

Variabili:

BLK	Numero dei blocchi interpretati. Se BLK=0, e' in uso il TIB come sorgente.
OFFSET	Puo' essere usato per spostare l'area degli schermi, cio' equivale a muoversi in un'area disco differente. E' meglio lasciarlo a zero.
PREV	Contiene l'indirizzo del buffer usato piu' di recente.
USE	Contiene l'indirizzo del buffer usato meno di recente.
SCR	Contiene il numero degli schermi in uso.

Funzioni:

+BUF	Rimuove TOS considerandolo l'indirizzo del buffer corrente e seleziona il prossimo buffer in sequenza, mettendo l'indirizzo di quest'ultimo in 2OS e un flag in TOS. Se il nuovo buffer e' quello identificato da PREV, il flag e' falso.
.LINE	Rimuove TOS considerandolo un numero di schermo. Se quel blocco e' contenuto in un buffer, l'indirizzo del buffer viene messo in TOS. Altrimenti il contenuto del buffer identificato da USE e' letto dal disco (in questo caso il disco RAM) e il blocco viene copiato in quel buffer, l'indirizzo del quale e' viene messo in TOS.
BLOCK	Rimuove TOS considerandolo il numero di un blocco. Se quel blocco e' contenuto in un buffer, l'indirizzo del buffer viene messo in TOS. Altrimenti il contenuto del buffer identificato da USE e' letto dal disco (in questo caso il disco RAM) e il blocco viene copiato in quel buffer, l'indirizzo del quale viene messo in TOS.
BUFFER	Rimuove TOS considerandolo il numero di un blocco ed assegna il prossimo buffer a quel blocco, salvando prima i contenuti del buffer su disco se sono stati aggiornati. L'indirizzo del buffer viene messo in TOS. (BLOCK usa BUFFER)
DRO	Mette OFFSET a zero.
EMPTY-BUFFERS	Mette i campi di controllo di tutti i buffers allo stato iniziale, cioe' vuoti.
FLUSH	Scrive su disco tutti i buffers modificati.
LINE	Rimuove TOS come numero di linea e mette in TOS l'indirizzo dell'inizio della linea nello schermo corrente.

R/W	Rimuove TOS considerandolo come un flag, 20S come numero di blocco e 30S come indirizzo. Se il flag e' falso, il dato e' scritto dal buffer al disco (il disco RAM). Se il flag e' vero viene eseguita una operazione di lettura.
TEXT	Viene rimosso TOS e considerato un delimitatore, il testo seguente e' copiato in PAD.
UPDATE	Il buffer identificato da PREV viene marcato "aggiornato" (updated), cioe' il suo contenuto e' stato alterato.

CAPITOLO V

Semplici Programmi

In questo capitolo verrano discussi ed esemplificati alcuni semplici programmi.

SEMPLICI PROGRAMMI

Armati ora di tutti le notizie che ci servono, possiamo cominciare a vedere alcuni semplici programmi. Per cominciare, cosa ne dite di un programma per archiviare dati memorizzati e visualizzarli?.

```
SCR # 1
  0 ( PROGRAMMA ARCHIVIO)
  1 : TASK ;
  2 : PLINE CR DUP 5 U.R 8 0 DO
  3         DUP C@ 3 .R 1+
  4         LOOP ;
  5 : PBLOCK CR 16 0 DO PLINE LOOP ;
  6 : GETN QUERY INTERPRET ;
  7 : DUMP HEX CLS ." Indirizzo di inizio ?"
  8        GETN BEGIN PBLOCK CR KEY DROP
  9        ?TERMINAL UNTIL ;
```

Mettete il programma nello schermo numero 1, controllatelo e, se necessario, correggete gli errori. Chiamate quindi FORTH 1 LOAD ; a questo punto, la parola DUMP chiamera' il programma. Nella maggior parte dei BASICs, sarebbe stato molto piu' lungo ottenere il formato richiesto ed eseguire la conversione esadecimale.

Il programma non e' impaginato nello stile formale del FORTH, ma comunque si carichera' perfettamente. Notate l'uso di TASK all'inizio, cosi' che il programma si potra' cancellare con FORGET TASK.

PLINE esegue un newline, quindi duplica il TOS che contiene l'indirizzo corrente dell'archivio. La copia viene usata per visualizzare l'indirizzo in un campo di cinque posizioni (5 U.R) e quindi viene inserito un ciclo a 8 iterazioni che duplica ancora l'indirizzo, legge il contenuto della locazione definita e visualizza il risultato in un campo di tre posizioni.

L'indirizzo, ancora in TOS, viene incrementato e il ciclo iterato. Il risultato finale e' una linea singola dell'archivio. E' in esadecimale poiche' DUMP stabilisce questa condizione.

PBLOCK ripete PLINE sedici volte, aggiungendo prima una linea vuota.

GETN mette l'indirizzo di partenza desiderato in TOS.

DUMP , la parola che richiama il programma, stabilisce il sistema esadecimale (HEX), cancella lo schermo e scrive un invito a inserire l'indirizzo di partenza. Questo sarebbe potuto essere incluso altrettanto bene in GETN per mostrare piu' chiaramente lo scopo di questa parola. Il ciclo BEGIN-UNTIL ripete il tutto finche' non viene premuto BREAK (CAPS SHIFT e 1), ma l'azione si interrompe ad ogni blocco finche' viene premuta un tasto. Notate che il risultato di KEY viene scartato (DROP) giacche' non interessa.

Mettete il programma nello schermo 1, controllatelo e, se necessario correggete gli errori. Quindi chiamate 1 LOAD dopodiche' la parola DUMP chiamera' il programma. Nella maggior parte dei BASICs sarebbe stato molto piu' lungo ottenere il formato richiesto ed eseguire la conversione esadecimale.

Potreste voler vedere il testo, piuttosto che i codici esadecimali. Benissimo, dobbiamo cambiare PLINE come segue:

```
: TPLINE CR DUP 5 U.R SPACE 16 0 DO
         DUP C@ 32 MAX DUP 160 > 1F 128 - ENDIF
         EMIT 1+ LOOP ;
```

Visto che e' archiviato un solo carattere per locazione, possiamo mettere 16 caratteri in una linea. Non vogliamo che vengano visualizzati codici sotto il 32, che potrebbero dare problemi, cosi' mettiamo 32 MAX in modo che venga preso 32 se il TOS antecedente era minore. I codici sopra 160 sono nell'area dei tokens, cosi', per loro, dobbiamo sottrarre 128. Usiamo EMIT per emettere il carattere del codice ASCII anziche' il valore esadecimale.

Queste due semplici routines, chiamano in causa un certo numero di punti importanti. Siccome abbiamo cambiato il nome di PLINE in TPLINE , questa non sara' piu' chiamata da PBLOCK per cui dobbiamo definire una nuova versione di PBLOCK chiamata TPBLOCK e una nuova versione di DUMP chiamata TDUMP . GETN non ha bisogno di modifiche se e' gia' stato complilato.

Se avessimo chiamato la definizione PLINE non avremmo avuto differenze poiche' PBLOCK avrebbe fatto riferimento al PLINE originale, quello che era stato definito prima di PBLOCK.

Usando l'EDITOR e' abbastanza semplice copiare la pagina 1 alla pagina 2 ed alterare i nomi come necessario, con la definizione di TPLINE in cima.

Il punto successivo riguarda i numeri. La compilazione e' stata fatta con il sistema allo stato decimale (DECIMAL), cosi' che i numeri sono stati dati in forma decimale. HEX non viene eseguito alla linea 7 dello schermo 1, poche' e' in compilazione. E' bene notare che LIST definisce il modo decimale per i propri scopi, cosi' e' generalmente meglio usare valori decimali nel codice sorgente. Quando e' piu' coveniente usare valori esadecimali, dovete solo inserire HEX, fuori da una

definizione in modo che venga eseguito e non compilato.

Cosa possiamo dire ancora? Ebbene, una rimarchevole assenza dal dizionario e' un qualsiasi tipo di generatore casuale, e la maggior parte dei programmi ne abbisognano prima o poi. Qui c'e'una possibilita':

```
0 VARIABLE SEED
: MODSEED SEED @ 75 U* 75 0 D+
          OVER OVER U< - - 1 - DUP SEED ! ;
: RND MODSEED U* SWAP DROP ;
: RAND -DUP 0= IF 23672 @ THEN SEED ! ;
```

RAND e' un generatore di numeri a caso. 0 RAND mettera' SEED al byte minore di FRAMES. Se il TOS non e' a zero, tale valore verra' messo in SEED.

RND e' l'effettivo generatore di numeri random dal punto di vista dell'utente. Viene usato nella forma n RND che genera un numero compreso fra 0 ed n.

MODSEED cambia il valore di SEED e crea il numero a caso di base fra 0 e 1 per il quale viene moltiplicato n.

Il lavoro effettivo delle parole e' meno importante del risultato che danno. Ci sono qui alcune definizioni che vi permettono di controllare la casualita':

```
0 VARIABLE ARRAY 62 ALLOT
: AGEN ARRAY SWAP 2 * + ;
: A! AGEN ! ;
: A@ AGEN @ ;
: ZERO 32 0 DO 0 I A! LOOP ;
: SETUP 0 RAND 1000 0 DO 32 RND DUP A@ 1+ SWAP A! LOOP ;
: DISP CR 32 0 DO I A@ 8 .R LOOP ;
: GRAPH CLS 32 0 DO I 8 * 0 PLOT I 8 * I A@ DRAW LOOP ;
: CHECK ZERO SETUP GRAPH ;
: DISCHECK ZERO SETUP DISP ;
```

Viene definito un array di 64 bytes. ZERO mette a zero l'array. SETUP genera quindi mille numeri casuali fra 0 e 32 (senza mai effettivamente raggiungere 32) e aggiunge 1 all'n-esimo elemento dell'array ogni volta che risulta il numero n. Sono disponibili due opzioni per controllare la casualita': DISCHECK chiama ZERO SETUP DISPLAY che mostrano i 32 numeri nell'array in quattro colonne, CHECK MOSTRA IL RISULTATO IN FORMA DI ISTOGRAMMA. E' possibile fare un controllo piu' rigoroso incrementando le dimensioni del ciclo DO in SETUP.

Questi semplici programmi meritano di essere analizzati in dettaglio; noterete che potete imparare da essi. Ci sono spesso modi alternativi di fare le stesse cose in FORTH. Potreste trovare che i manipolatori dello stack meritano di essere riesaminati. Provare a combinare le routines GRAPH e DISP per ottenere un grafico con i numeri.

Dopo tutto, sperimentate qualche routine per conto vostro, se siete a corto di idee, prendete un programma BASIC piuttosto

semplice e convertitelo in FORTH. Se l'originale e' disordinato, dovrete prima ordinarlo, giacche' i programmi FORTH devono essere strutturati. Questo significa che devono essere stesi in maniera coerente. In particolare le definizioni devono essere nel giusto ordine.

Nel prossimo paragrafo, esamineremo l'approccio alla scrittura di un programma piu' grande di quelli gia' visti.

PROGETTAZIONE DI UN PROGRAMMA

Verra' usato il problema classico della "Torre di Hanoi" per illustrare il modo in cui va progettato un programma. Il problema coinvolge un certo numero di dischi di dimensioni differenti arrangiati in tre pile. Iniziando con tutti i dischi nella pila 0 , questi devono essere mossi, uno alla volta, alla pila 2, non potendosi mai porre un disco sopra uno di dimensioni minori.

La struttura complessiva del programma sara' approssimativamente:

```
Visualizza il titolo
Definisci le condizioni iniziali
Muovi i dischi
Chiedi se si vuole ripetere
Se e' cosi',ripeti dalla inizializzazione
```

Il primo passo, e' decidere il formato del display. Sara' conveniente porre i centri delle pile nella sesta, diciassettesima e ventottesima colonna. Per il plotting grafico, le coordinate saranno approssimativamente 43, 131 e 219. Approssimativamente, poiche' per averli nel centro dei numeri delle pile sarebbe necessario creare caratteri speciali per i numeri, visto che i numeri normali non hanno un punto in posizione centrale.

Se stabiliamo che l'n-esimo disco sia grande 6*n punti, potremo maneggiare fino a 12 dischi.

Deve poi essere deciso il metodo di lavoro. Il diagramma a blocchi che viene presentato mostra un approccio che non usa ne' ricorsioni ne' formule empiriche ed e' giustificabile sul piano della semplice logica.

Ogni mossa viene specificata da tre numeri nello stack. TOS da' il numero del disco, e verra' mostrato come n; 2OS contiene il numero della pila dalla quale deve essere preso il disco e verra' mostrato come s; 3OS contiene il numero della pila in cui va messo il disco e verra' mostrato come d.

Se devono essere mossi n dischi, i dati iniziali nello stack devono essere 2 0 n , chidendo la mossa del disco n dalla sorgente 0 alla destinazione 2. Questa mossa puo' essere fatta solo se tutti gli altri dischi sono nella pila 1, e la funzione GEN1 calcola 1 0 n-1 come mossa precedente. Questa, tuttavia, richiede 2 0 n-2 come mossa precedente e cosi' via. GEN1 prende i tre dati originali nello stack d s e n e ne crea altri tre che

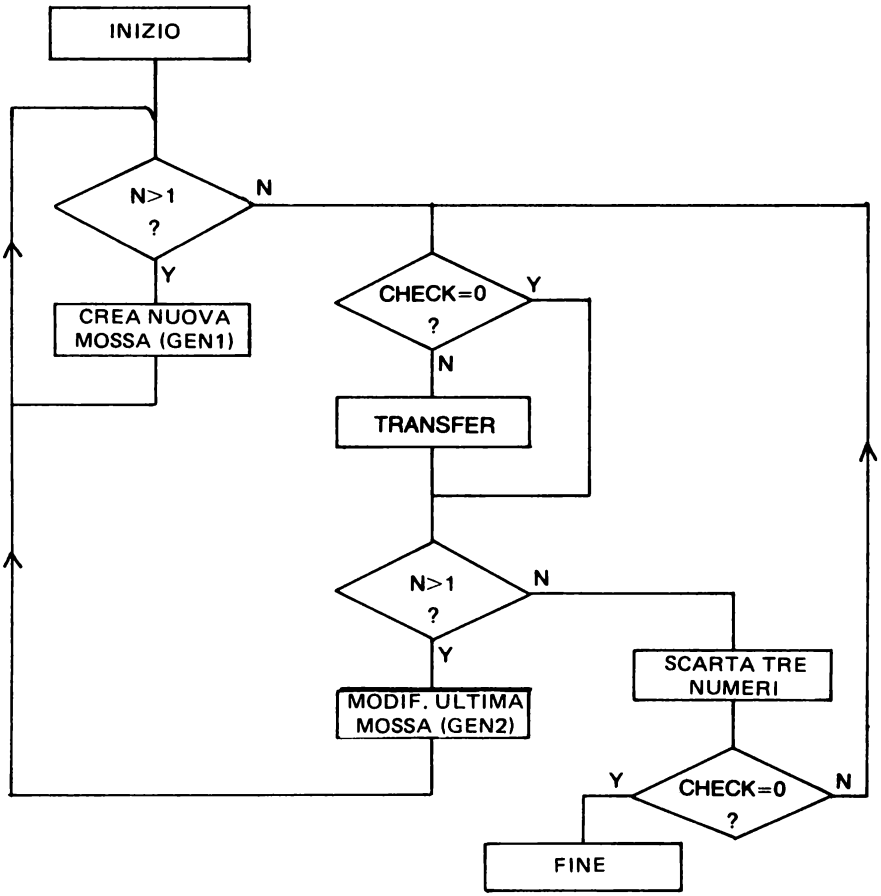


DIAGRAMMA A BLOCCHI DELLA FUNZIONE MOVE DELLA TORRE DI HANOI

sono collegati ai precedenti come segue:

Il nuovo 3OS viene calcolato come: 3-d-s.
Il 2OS rimane inalterato.
Il TOS viene decrementato.

Quando la ripetizione riduce il TOS a 1, il processo effettivo delle mosse puo' iniziare. Se ci sono cinque dischi, lo stack contiene:

```
2 0 5 1 0 4 2 0 3 1 0 2 2 0 1
```

Le mosse 2 0 1 e 1 0 2 possono essere fatte, mettendo il disco 1 nella pila 2 e il disco 2 nella pila 1. Prima che possa essere fatta la terza mossa, 2 0 3, occorre una mossa extra per cancellare la pila 2. Questa e' calcolata da GEN2, che modifica una mossa esistente, piuttosto che aggiungerne una nuova. Il cambiamento e':

Il nuovo 3OS e' uguale a d, il 3OS originale.
Il nuovo 2OS viene calcolato come 3-d-s. Siccome il totale di tutti i numeri delle pile e' 3, questo identifichera' la pila non coinvolta nella mossa originale.
Il nuovo TOS e' uno meno del TOS originale.

Questo, nel contesto precedente, da' 1 2 1, muovendo il disco 1 dalla pila 2 alla pila 1.

Se viene calcolata una sequenza di mosse su questa base, si notera' che le mosse generate sono quelle richieste per fare la mossa specificata. Notate, tuttavia, che n puo' avere due significati. Puo' significare disco n o n dischi. La distinzione non e' importante in pratica.

GEN1 sara' ovviamente una parola FORTH. Dovra' prima copiare 3OS, e sara' utile a questo proposito una subparola speciale:

```
: 3OVER >R OVER R> SWAP ;
```

TOS viene passato a TORS mentre OVER copia il 3OS come nuovo TOS. Dopo aver ristabilito il TOS originale, SWAP porta la copia di 3OS in 2OS. Lo stack originale d s n diventa d s n d.

Puo' allora essere definito GEN1:

```
: GEN1 3OVER 3 SWAP - 3OVER - 3OVER 3OVER 1 - ;
```

Come tutte le nuove parole di una certa complessita', dovrebbe essere controllata tabulando i cambiamenti dello stack:

```
              Stack
              d s n
3OVER         d s n d
3 SWAP -      d s n 3-d
3OVER -       d s n 3-d-s
3OVER         d s n 3-d-s s
3OVER         d s n 3-d-s s
1 -           d s n 3-d-s s n-1
```

I tre nuovi dati sono stati aggiunti senza modificare il resto dello stack.

GEN2 puo' essere definita analogamente come:

```
: GEN2 ROT DUP >R 3 SWAP - ROT - R> SWAP ROT 1 - ;
```

Controllando i cambiamenti dello stack:

```
                Stack
                d s n
   ROT          s n d
   DUP >R       s n d          TORS=d
   3 SWAP -     s n 3-d
   ROT          n 3-d s
   -            n 3-d-s
   R>           n 3-d-s d
   SWAP         n d 3-d-s
   ROT          d 3-d-s n
   1 -          d 3-d-s n-1
```

Dobbiamo ora considerare la funzione TRANSFER . Deve cancellare il disco n dalla sua posizione attuale e visualizzarlo nella nuova posizione. Nel processo, lo stack non deve essere alterato, giacche' GEN2 potrebbe aver bisogno dei dati. Se non e' coinvolto GEN2, i tre dati che definiscono la mossa saranno scartati. Sarebbe senza dubbio possibile ottenere tale risultato usando solo lo stack, ma sara' piu' semplice far uso di alcune variabili e di un array. Le variabili sono DSIZE (meta' larghezza del disco in punti), XPOS (centro verticale della colonna del disco in coordinate di punti), YPOS (coordinata verticale del fondo del disco).

L'array e' PILE, e contiene il numero di dischi in ogni pila. Poiche' c'e' solo un array, la definiremo come segue:

```
0 VARIABLE ARRAY PILE 4 ALLOT
: AGET PILE SWAP 2 * + ;
: A! AGET ! ;
: A@ AGET @ ;
```

L'indirizzo dell'elemento P sara' messo in TOS da P AGET, e si puo' accedere al contenuto dell'elemento con P A@ . Potremo scrivere nell'elemento con X P A! dove X e' il dato da scrivere. Siccome i numeri coinvolti sono piccoli, aremmo potuto usare un array a tre bytes anziche' un array a tre parole, ma la questione non e' molto importante.

Usando la funzione BASIC OVER, possiamo usare la stessa routine per disegnare e cancellare i dischi:

```
: DDRAW 1 OVER 6 0 DO     (Disegna sei linee orizzontali)
  XPOS @ DSIZE @ -        (x in PLOT = XPOS-DSIZE)
  YPOS @ I + PLOT         (y in PLOT = YPOS+I)
  XPOS @ DSIZE @ +        (x in DRAW = XPOS+DSIZE)
  YPOS @ I + DRAW         (y in DRAW = YPOS+I)
  LOOP 0 OVER ;
```

Abbiamo ora bisogno di una routine che traduca i tre dati in cima allo stack nei valori richiesti delle variabili:

```
: SETUP OVER OVER   (Stack d s n in d s n s n )
  3 * DSIZE !       (DSIZE = 3*n)
  DUP A@            (Stack d s n s PILE(s))
  1 + 8 * YPOS !    (YPOS = 8 * (PILE(s) + 1))
  88 * 43 + XPOS !  (XPOS = s * 88 + 43)
  DDRAW ;
```

Cio' cancellerebbe il disco in cima alla pila s. Per inserire lo stesso disco nella pila d, e deve prima aggiustare il

contenuto dell'array PILE per tener conto della mossa, e quindi chiamare ancora SETUP con lo stack modificato temporaneamente da d s n in d s d n ;

```
: ADJUST OVER AGET - 1 SWAP + !  (decrementa PILE(s))
  3OVER AGET 1 SWAP + ! ;        (incrementa PILE(d))
: TRANSFER SETUP ADJUST 3OVER
  SWAP SETUP SWAP DROP ;
```

Alla fine, per completare il diagramma a blocchi, abbiamo bisogno di:

```
: CHECK 0 A@ 1 A@ + ; (ritorna 0 quando la mossa e'
                       completa)
```

La parola MOVE che copre il diagrmma a blocchi puo' ora essere definita:

```
: MOVE BEGIN CHECK WHILE
       BEGIN DUP 1 > WHILE
        GEN1
       REPEAT
       BEGIN
        CHECK IF TRANSFER ENDIF
        DUP 1 > NOT WHILE
        DROP DROP DROP
       REPEAT GEN2
      REPEAT  ;
```

Confrontatela con il diagramma a blocchi.

Abbiamo ora bisogno di INIT , che ha la forma:

```
: INIT ENQUIRE          (Chiede quanti dischi)
  LINEDRAW              (Disegna la linea di base)
  PILEDRAW              (Disegna la pila iniziale)
  WAIT ;                (Pausa prima dell'inizio di MOVE)
: ENQUIRE 2 0           (Destinazione e sorgente iniziale)
  CLS 10 5 AT           (Cancella lo schermo, posiziona il
  ." Quanti dischi ?"    testo)
  QUERY INTERPRET       (Ottiene la risposta)
  2 MAX 12 MIN ;        (Limiti dei dischi)
: LINEDRAW 1 INK        (Linea blu)
  CLS 20 0 AT           (Cancella schemo, posiziona linea)
  ." (5 SPIV) 1 (10 SPIV) 2 (10 SPIV) 3 (4 SPIV)"
  2 INK ;               (Dischi rossi)
```

Quanto a sopra, 10 SPIV significa 10 Spazi in Inverse Video, Graphic 8.

```
: PILEDRAW 3 0 DO 0 I A! LOOP (Azzera elementi array)
  OVER OVER DUP 0 DO          (Stack d s n in d s n s n )
  0 AGET 1 SWAP +!            (Incrementa PILE(0))
  I - SETUP DROP OVER         (Disegna disco n-1. Ripr. n)
  LOOP DROP DROP ;            (Scarta dati non rilevanti)
```

Abbiamo bisogno di una funzione per rispondere alla richiesta di far partire ancora il programma:

```
: ASK 0 0 AT ." Ancora?" KEY 89 - ;
```

Ritornera' 0 se viene premuto Y.

Abbiamo bisogno anche di una funzione che mostri il titolo:

```
: TITLE CLS 10 6 AT ." LA TORRE DI HANOI" WAIT ;
```

Tutto cio' che ci rimane da fare e' definire la funzione al livello piu' alto.

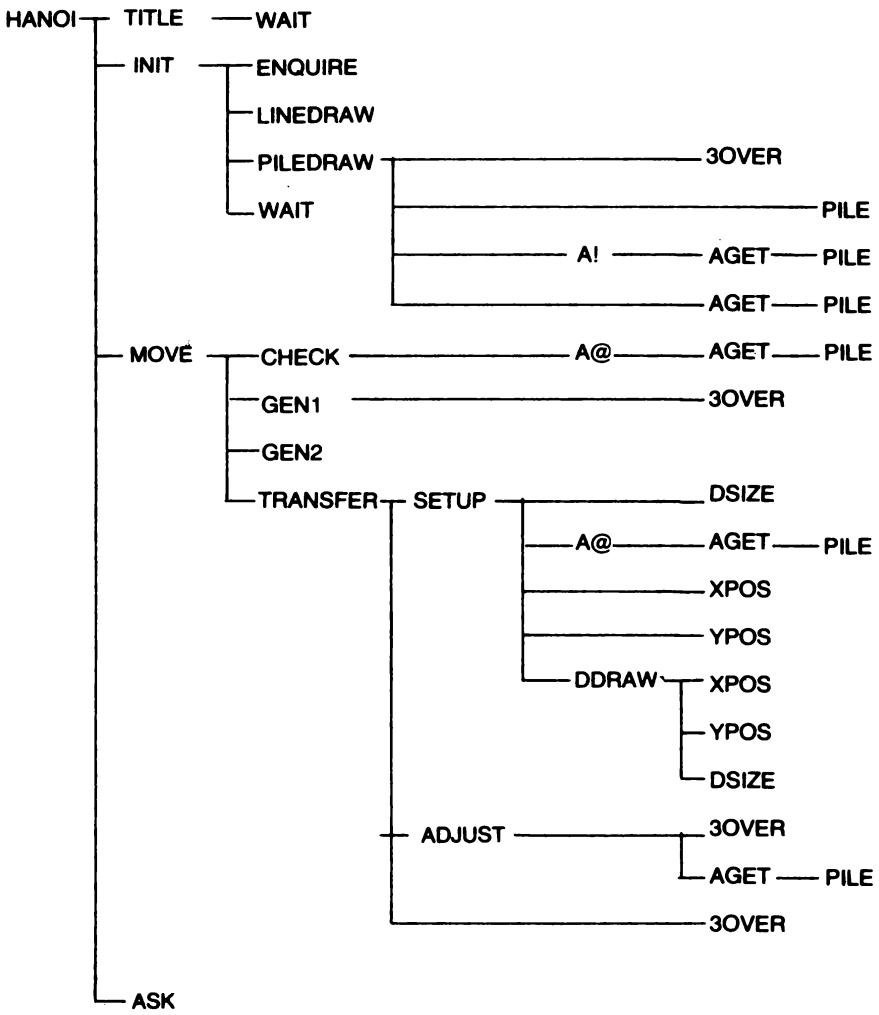
```
: HANOI TITLE BEGIN INIT MOVE 0 INK ASK UNTIL ;
```

Possiamo ora costruire una gerarchia di funzioni che mostri come ognuna sia in relazione alle altre, e questa sara' la base per decidere in quale ordine andranno compilate:

Le definizioni alla colonna destra andranno definite prima, quindi colonna per colonna verso sinistra. L'ordine non e' completamente rigido, provvedendo che le priorita' necessarie vengano osservate. Nel listato allegato viene mostrata una possibile stesura in quattro schermi.

Questo e' un programma moderatamente complesso, principalmente a causa dell'utilizzazione dello stack per muovere i dati. E' stato scelto perche' illustra un'ampia gamma di tecniche, senza diventare troppo astruso.

E' sempre consigliabile salvare gli schermi su nastro prima di far correre un tale programma. Un piccolo errore nella battitura puo' causare il blocco del sistema o qualche altra disfunzione, e tutti i dati memorizzati possono allora andare persi.


HANOI
TITLE
WAIT
INIT
ENQUIRE
LINEDRAW
PILEDRAW
3OVER
PILE
A!
AGET
PILE
AGET
PILE
WAIT
MOVE
CHECK
A@
AGET
PILE
GEN1
3OVER
GEN2
TRANSFER
SETUP
DSIZE
A@
AGET
PILE
XPOS
YPOS
DDRAW
XPOS
YPOS
DSIZE
ADJUST
3OVER
AGET
PILE
3OVER
ASK

```
SCR # 1
  0 ( LA TORRE DI HANOI: 1)
  1 : TASK ;
  2 0 VARIABLE DSIZE
  3 0 VARIABLE XPOS
  4 0 VARIABLE YPOS
  5 0 VARIABLE PILE 4 ALLOT
  6 : 3OVER >R OVER R> SWAP ;
  7 : AGET PILE SWAP 2 * + ;
  8 : A! AGET ! ;
  9 : A@ AGET @ ;
 10 : DDRAW 1 GOVER 6 0 DO
 11         XPOS @ DSIZE @ -
 12         YPOS @ I + PLOT
 13         XPOS @ DSIZE @ +
 14         YPOS @ I + DRAW
 15         LOOP 0 GOVER ; -->

SCR # 2
  0 ( LA TORRE DI HANOI: 2)
  1 : SETUP OVER OVER
  2           3 * DSIZE !
  3           DUP A@
  4           1 + 8 * YPOS !
  5           88 * 43 + XPOS !
  6           DDRAW ;
  7 : ADJUST OVER AGET -1 SWAP +!
  8            3OVER AGET 1 SWAP +! ;
  9 : WAIT 20000 0 DO LOOP ;
 10 : ENQUIRE 2 0 CLS 10 5 AT
 11            ." Quanti dischi ?"
 12            QUERY INTERPRET
 13            2 MAX 12 MIN ;
 14 : CHECK 0 A@ 1 A@ + ;
 15 -->

SCR # 3
  0 ( LA TORRE DI HANOI: 3)
  1 : LINEDRAW 1 INK CLS 20 0 AT
  2 ." _____1_________2_________3____" 2 INK ;
  3 : PILEDRAW 3 0 DO 0 I A! LOOP
```

```
 4     OVER OVER DUP 0 DO
 5     0 AGET 1 SWAP +!
 6     I - SETUP DROP OVER
 7     LOOP DROP DROP ;
 8 : GEN1 3OVER 3 SWAP - 3OVER
 9           - 3OVER 3OVER 1 - ;
10 : GEN2 ROT DUP >R 3 SWAP -
11    ROT - R> SWAP ROT 1 - ;
12 : TRANSFER SETUP ADJUST 3OVER
13    SWAP SETUP SWAP DROP ;
14 : INIT ENQUIRE LINEDRAW
15          PILEDRAW WAIT ; -->

SCR # 4
 0 ( LA TORRE DI HANOI: 4)
 1 : TITLE CLS 10 0 AT
 2 ." LA TORRE DI HANOI" WAIT ;
 3 : MOVE BEGIN CHECK WHILE
 4            BEGIN DUP 1 > WHILE
 5            GEN1
 6           REPEAT
 7           BEGIN
 8            CHECK IF TRANSFER ENDIF
 9            DUP 1 > NOT WHILE
10          DROP DROP DROP
11         REPEAT GEN2
12       REPEAT ;
13 : ASK 0 0 AT ." Ancora ?" KEY 89 - ;
14 : HANOI TITLE BEGIN INIT
15    MOVE 0 INK ASK UNTIL ;
```

CAPITOLO VI

La compilazione

Questo capitolo tratta delle parole che vengono usate nella compilazione, e illustra alcuni dei modi meno ovvi nei quali tali parole possono essere usate.

LA COMPILAZIONE

Quando il sistema e' in modo compilazione, la maggior parte delle parole che trova nel corso dell'input vengono incorporate nella nuova parola che entra nel dizionario, e l'indirizzo del suo campo codice viene aggiunto al nuovo campo parametri. Alcune parole, tuttavia, sono trattate in modo differente.

Le parole che hanno il "bit precedenza" nel byte lunghezza in condizione di verita', non sono compilate, vengono eseguite. Per esempio, la definizione della parola DO e':

```
COMPILE (DO)
HERE
3
```

La prima linea inserisce l'indirizzo del campo codici per (DO) nel campo parametri e HERE quindi salva il contenuto del puntatore del dizionario per un uso successivo, essendo il contenuto presente nello stack. Quindi viene messo 3 nello stack, essendo il numero di controllo per la combinazione DO-LOOP.

La definizione di LOOP e':

```
3 ?PAIRS
COMPILE (LOOP)
BACK
```

Viene messo un altro 3 nello stack e ?PAIRS lo confronta con 2OS, che dovrebbe contenere il 3 lasciato da DO. Se non e' cosi', allora e' stato fatto qualche errore, ad esempio nel bilanciare un altro tipo di ciclo.

Se la comparazione mostra una corrispondenza, l'indirizzo del campo codici per (LOOP) viene aggiunto al nuovo campo parametri e quindi viene chiamato BACK. Questi ancora chiama HERE per mettere il contenuto del puntatore del dizionario nello stack, ed esegue una sottrazione. Il risultato viene scritto nel nuovo campo parametri, essendo l'ampiezza del salto necessario a raggiungere un punto immediatamente dopo il (DO).

Procedimenti simili vengono applicati alle altre combinazioni di ramificazione e di ricorsione.

Anche due punti e punto e virgola hanno il bit precedenza vero. I due punti, prima controllano se il sistema si trova nel modo esecuzione diretta, giacche' non sono ammessi nel modo compilazione. Quindi il puntatore dello stack corrente viene salvato in CSP (Current Stack Pointer) e CONTEXT viene messo uguale a CURRENT per assicurarsi che venga usato il valore esatto del puntatore del dizionario, mettendo la nuova parola nel vocabolario selezionato. CREATE definisce quindi i primi due campi della nuova parola. Il campo codici viene fatto puntare a

una corta routine in linguaggio macchina che stabilisce le condizioni per l'interpretazione del campo parametri, salvando il valore del puntatore dell'istruzione corrente nel Return Stack e mettendo il puntatore ad un nuovo valore, l'inizio del campo parametri. Il puntatore dell'istruzione viene usato per leggere l'indirizzo associato e altri dati necessari all'esecuzione.

Il punto e virgola fa il processo inverso. Si aspetta che il puntatore dello stack sia tornato al valore salvato in CSP dai due punti e genera un messaggio d'errore se non e' cosi'. Quindi viene compilata ;S nell'ultima posizione del campo parametri, il bit "macchia" viene posto nello stato valido e viene ripresa l'esecuzione diretta. Quando viene eseguito, ;S rimuove l'indirzzo associato di ritorno dal Return Stack e lo mette nel puntatore dell'istruzione.

Per la maggior parte degli scopi, non e' necessario ricordare tali processi ma questi possono essere importanti se usate trucchi per la definizione di nuove parole. Alcune delle definizioni date nell'Appendice A sembrano abbastanza semplici, ma se tentate di compilarle direttamente noterete il sitema fa obiezioni.

Considerate, per esempio, le parole formate da parentesi quadre. Il processo dei due punti usa] per abilitare il modo compilazione e il punto e virgola usa [per ritornare all'esecuzione diretta. Tutto cio' che fanno queste parole e' di cambiare il valore della variabile STATE , ma sono comunque molto utili, permettendo un interludio all'esecuzione in diretta nel mezzo di un processo di compilazione. Cio' potrebbe venir usato per calcolare un valore critico che dipende da una parola compilata precedentemente.

I due punti usano anche ;CODE, che compila (;CODE). Quando viene eseguito, (;CODE) fa puntare, al campo codici della parola compilata piu' di recente, a una routine in codice macchina che segue ;CODE . Nel caso dei due punti, l'effettivo codice macchina viene chiamato da tutte le definizioni che hanno un reale campo parametri. Se fosse disponibile un assembler, verrebbe inserito automaticamente per definire il codice in questione. In sua assenza possono essere usate le parole , e C,. La virgola memorizza TOS nella prossima locazione del dizionario, mentre C, memorizza solo il byte basso di TOS. Usando queste due parole sarebbe possibile definire una nuova parola direttamente, ma sarebbe un notevole spreco di tempo e di fatica. Tuttavia, e' a volte utile usare le parole per definire un dato od un'associazione specifica all'interno di un campo parametri, o un dato all'interno di un array. In quest'ultimo caso non e' necessario usare ALLOT per far posto all'array poiche' la virgola e C, avanzano entrambi il puntatore del dizionario alla prima locazione vuota.

Forse la parola piu' bizzarra nel FORTH e' quella con un campo nome che contiene C1 80 . Lunghezza di una lettera, in altre parole, tale lettera ci viene resa nota da 80H-80H=0 . Viene usato un codice nullo per terminare gli schermi e i buffers, e per fermare l'interpretazione. Se viene

accidentalmente inserito un codice nullo nel mezzo di uno schermo, sara' impossibile interpretare lo schermo oltre quel punto.

LATEST mette nello stack l'indirizzo del campo nome della parola del dizionario definita piu' di recente. Questa viene usata da IMMEDIATE , che definisce il bit precedenza per quella parola, permettendo di definire quelle parole che saranno eseguite nel modo compilazione.

D'altro canto, [COMPILE] causera' la compilazione della parola seguente anche se ha vero il bit precedenza. Le possibilita' aperte da questa parola possono disorientare, in principio. E' possibile visulizzare le parole che definiranno i cicli DO o le altre funzioni di ramificazione e ricorsione. Tuttavia, l'esempio classico e':

```
: XXXX [COMPILE] FORTH ;
```

Senza [COMPILE], la parola FORTH sarebbe eseguita quando fosse compilato XXXX, lasciando XXXX inattiva. Poiche' c'e', la definizione tramite due punti sara' equivalente a FORTH e l'esecuzione di XXXX selezionera' il vocabolario FORTH. In un senso, [COMPILE] rimanda l'azione della parola sulla quale agisce, rendendola effettiva al momento dell'esecuzione, piuttosto che durante la compilazione.

Poi abbiamo due parole importanti che raramente vengono esaminate, LITERAL e DLITERAL. Vengono chiamate quando INTERPRET trova un numero nell'input. Nel modo diretto, non fanno niente, giacche' il numero viene messo nello stack e li' rimane. Nel modo compilazione, LITERAL compila LIT e poi usa la virgola per trasferire il numero dallo stack al campo parametri. Quando il campo viene interpretato, LIT trasferisce il numero di nuovo nello stack. DLITERAL agisce in modo simile ma compila LIT , quindi TOS, ancora LIT , poi 2OS. Come suggerisce la D, e' usata per trattare numeri doppi. CREATE definisce i primi due campi di una nuova parola, usando WIDTH per controllare che la lunghezza della parola sia entro limiti validi. WIDTH e' una variabile, generalmente posta a 31.

La parola ' e' utile a volte nella forma:

```
' XXXX
```

nell'esecuzione diretta, mette l'indirizzo del campo parametri di XXXX nello stack. Nella compilazione, trasferisce l'indizizzo come fosse la parola LIT inserita nel nuovo campo parametri.

A questo punto e' necessario chiamare un'interruzione. Le parole che sono piu' di frequente usate in modo diretto nel processo di compilazione sono gia' state descritte, con un'eccezione, che e' abbastanza complessa da meritare un paragrafo a parte.

Un fascino e una frustrazione del FORTH e' che ci sono molti trucchi sottili ed esoterici con i quali potete giocare, cosi' tanti che non avrebbe senso tentare di spiegarli tutti. I programmi forniti in questo libro, danno alcuni esempi, ma senza l'intenzione di coprire l'intera gamma di possibilita'. Dovete inventare voi stessi le vostre soluzioni, visto che probabilmente esse sono volte a risolvere problemi che non sono abbastanza comuni da avere risposte gia' pronte.

<BUILDS . . .DOES>

E' stato detto che la combinazione <BUILD...DOES> e', sia una delle caratteristiche piu' potenti del fig-FORTH, che una delle piu' difficili da spiegare. E' uno strumento usato per eseguire processi complessi che creano altri strumenti. Ecco un esempio:

```
: ARRAY <BUILDS 2 * ALLOT DOES> SWAP 1 - 2 * + ;
```

Questo crea una nuova parola, ARRAY, che puo' essere usata come segue:

```
8 ARRAY HEAP
```

Questo creera' un'array con spazio per nove parole a due bytes, l'8 specificato piu' uno, forma la variabile di base, come nel semplice metodo di creazioni degli array gia' discusso. Tutto questo e' ottenuto con 2 * ALLOT che segue <BUILDS , ma c'e' di piu'. Se chiamiamo 5 HEAP , le parole che seguono DOES> decrementeranno 5 a 4, raddoppieranno il risultato e aggiungeranno l'indirizzo di base dell'array. Cosi' n HEAP @ leggera' l'ennesimo elemento dell'array, n HEAP ? visualizzera' tale elemento e m n HEAP ! mettera' l'elemento a m.

La parola ARRAY rimane disponibile come strumento per creare ulteriori arrays in modo simile. Si sara' notato che questo risparmia un mucchio di problemi rispetto al metodo illustrato in precedenza.

Il concetto da afferrare e' che le parole che seguono <BUILDS vengono usate per determinare la forma di base della nuova parola, mentre le parole che seguono DOES> determinano il processo che sara' ad essa associato.

Ecco un esempio piu' complesso che gestisce arrays bidimensionali e controlla gli indici per assicurarsi che non siano fuori dai limiti definiti dalle dimensioni date. Vengono coinvolte tre nuove parole, e la compilazione e' eseguita con selezionata la notazione esadecimale per ragioni di convenienza e di chiarezza.

```
HEX
: OFB ." Fuori dai limiti dell'array. " SP! QUIT ;
: OFBCK >R ROT ROT DUP R> DUP FF AND ROT < IF OFB ENDIF
100 / >R OVER R> DUP ROT < IF OFB ENDIF ;
: ARRAY2 <BUILDS FF AND DUP C, SWAP FF AND DUP C, * 2 *
ALLOT DOES> DUP @ OFBCK SWAP 1 - * + + ;
```

Con queste parole definite viene creato un'array con le dimensioni x e y tramite:

```
x y ARRAY2 MATRIX
```

Le dimensioni vengono memorizzate all'inizio dell'array e il numero di bytes riservato e' 2*x*y. Tutto cio' viene determinato dalle parole che seguono <BUILDS. Le parole che seguono DOES> hanno effetto quando viene eseguita MATRIX. Notate che x e y vengono memorizzate da C, e dunque occupano un byte ognuna. DUP @ le mette entrambe in TOS , preservando l'indice dietro di loro, OFBCK viene quindi chiamato per confrontare gli indici con x e y e per avvertire se sono fuori dai limiti. Trovate i cambiamenti dello stack, ma ricordate che viene usata la notazione esadecimale cosicche' una divisione per 100 e' in

realta' una divisione per 256, che porta il byte superiore di TOS nella posizione del byte inferiore.

Questi esempi illustrano l'uso di <BUILD...DOES> in un contesto particolare, ma il concetto generale dovrebbe essere abbastanza chiaro. Parlando in generale, <BUILD stabilisce una costante, e il codice seguente estende la compilazione. DOES> usa un segmento standard di codice macchina per eseguire un piccolo gioco di prestigio cosi' che le parole che seguono siano aggiunte alla parola creata con <BUILDS.

Ci sono altri modi di creare strumenti di compilazione, ma il modo migliore di impararli e' di tentare varie combinazioni e di controllare le definizioni che risultano.

LEGGERE IL DIZIONARIO

E' a volte utile poter vedere come sono state inserite nel dizionario le parole che abbiamo definito. Per cio', c'e' un semplice programma:

```
: SCAN BEGIN CR
     DUP @ 2+ NFA ID.
     KEY CASE
     68 OF 2+ DUP @ U. 2+ 0 ENDOF
     69 OF 1 ENDOF
     70 OF 2+ 0 ENDOF
     ENDCASE UNTIL ;
: $ -FIND QUERY INTERPRET
     IF DROP CLS SCAN
     ELSE ." NON TROVATA " ENDIF ;
```

Inserendo $ seguito, dopo uno spazio, dal nome della definizione che volete esaminare e premete return due volte. Il primo nome apparira' sullo schermo in alto. Premete F per avere il prossimo nome, a meno che il primo sia BRANCH, 0BRANCH, LIT, <LOOP> o <+LOOP>. A tutti questi seguono parole di dati, e questi e le prossime parole possono essere ottenute premendo D. Per uscire premete E. La fine della definizione e' in genere marcata ;S . Se continuate dopo di questa, vi possono succedere le cose piu' strane.

Sarebbe simpatico, naturalmente, far chiamare a nomi speciali le azioni appropriate, cosi' che sia tutto automatico, ma ci sarebbe ancora un problema con (.") , che e' seguito da testo. Per continuare, sarebbe necessario fare una scansione attraverso il testo, che non sarebbe troppo difficile. Se volete usare parecchio la routine, dovreste essere in grado di trovare il modo di renderla piu' automatica.

Considerando l'ampiezza dei salti, ricordate che rappresentano un numero di bytes, e ogni parola o dato associato occupa due bytes. Una ampiezza di dieci, andra' dunque avanti di cinque dati.

Una piccola stranezza. E' necessario caricare i programmi di cui sopra in decimale, poiche' ci sono i codici ASCII 68, 69, e 70 in tale forma. Il programma, d'altra parte e' meglio che sia eseguito in esadecimale (HEX). Come vi assicurereste che vengano rispettati questi requisiti?.

ANCORA SULLA COMPILAZIONE

Il sistema normale di compilazione tiene automaticamente conto dei progressi della compilazione, e quando provate metodi meno diretti dovete essere preparati a fare altrettanto. Dovete esaminare la struttura dettagliata delle parole che usate, e questa e' una delle ragioni per le quali e' stata fornita l'Appendice A.

Nella sezione dell'appendice che copre il vocabolario EDITOR, troverete una piccola stranezza. Le parole R e I sono state ridefinite, ma le definizioni che seguono hanno il significato precedente. Per ottenere questo tipo di trucchi e' solamente necessario inserire la parola FORTH in un punto adatto, dopodiche' verra' compilata la versione FORTH della parola. La parola EDITOR assicurera' piu' tardi che la definizione continui sulla base delle parole EDITOR gia' definite.

Per definire un vocabolario speciale dovete prima stabilire la parola SPECIAL (o in qualsiasi modo lo vogliate chiamare) inserendo:

```
VOCABULARY SPECIAL
```

che inserisce una parola nel vocabolario FORTH. Per selezionare il vocabolario SPECIAL dovete fare:

```
SPECIAL DEFINITIONS
```

Le parole definite d'ora in poi verranno messe nel vocabolario SPECIAL.

Perche' avere un vocabolario speciale? Una buona ragione e' che un vocabolario troppo grande rallenta la compilazione, ed accedere ad un vocabolario specializzato puo' essere molto piu' veloce. Potreste anche voler usare le stesse parole per altri scopi, come nell'EDITOR.

Avendo compilato un programma, potreste trovar noioso il fatto che questi debba essere ricompilato ogniqualvolta lo volete eseguire. Non c'e' bisogno di questo. Potete savare l'intero dizionario esteso, includendo il vocabolario originale, in una operazione.

Per far cio', dovete prima sapere quanto salvare, e SIZE vi dara' tale informazione. Dovete poi alterare le locazioni che forniscono i dati di inizializzazione, in modo che si tenga conto dell'estensione del dizionario.

Il primo passo e' assicurarsi che non ci si trovi in un vocabolario speciale, inserendo:

```
FORTH DEFINITIONS
```

Poi, e' una questione di gusti, mi sembra piu' facile definire HEX, visto che stiamo trattando di indirizzi e spiazzamenti, che di solito sono piu' familiari in esadecimale.

LATEST contiene l'indirizzo del campo nome dell'ultima parola definita nel dizionario, e tutte le ricerche iniziano da quell'indirizzo, che deve essere posto in 5E4C:

```
LATEST 0C + ORIGIN !
```

Poi il puntatore del dizionario deve essere copiato in 5E5C e 5E5E:

```
HERE 1C + ORIGIN !
HERE 1E + ORIGIN !
```

Se volete proteggere i vostri programmi dovete muovere FENCE alla stessa posizione:

```
HERE FENCE !
```

Per finire dobbiamo fare:

```
' FORTH 8 + 20 + ORIGIN !
```

Questo mette il PFA della parola FORTH,piu' 8, in 5E60.

Il programma emendato puo' essere dunque salvato usando la linea 9 del rudimentale programma BASIC, opportunamente alterato per tener conto di SIZE. Dopo la verifica, il programma e' pronto per essere caricato al posto del nastro FORTH originale, il quale dovrebbe comportarsi come faceva prima di essere salvato.

Questa tecnica e' particolarmente utile se avete creato vostre estensioni del FORTH per uso generale.

CAPITOLO VII:

Tecniche di Programmazione

Questo capitolo tratta di alcune delle tecniche piu' complesse che possono essere utilizzate nella stesura dei programmi.

ARITMETICA NON INTERA

Se trovate troppo limitante l'aritmetica intera, guardate la definizione di DRAW nell'Appendice A (Parola del Dizionario No. 447). Essa lavora con incrementi di 0.0000152.

Il metodo usato consiste mel trattare un numero doppio come se fosse scalato di un fattore 1/65536. Per esempio, la parola LASTY viene letta dalla parola BASIC spazio ed e' quindi convertita in un numero doppio con l'aggiunta di una parola inferiore uguale a zero - no, non una parola superiore, una parola inferiore. In termini interi, il risultato e' LASTY*65536 che viene memorizzato nella variabile Y1. La versione scalata di LASTX e' analogamente memorizzata in X1.

Le coordinate di posizione richieste, vengono quindi confrontate con LASTY e LASTX nella loro forma non modificata per determinare quanti passi sono necessari per disegnare la linea. Sara' o ABS(X-LASTX) o ABS(Y-LASTY), il piu' grande dei due. L'incremento nominale dei valori di X e Y viene calcolato dividendo 65536 volte i cambiamenti totali per il numero di passi. Viene usato M/MOD , con vie alternative per tener conto di differenze positive e negative, e il risultato viene memorizzato in INCX e INCY come numeri doppi.

Siccome i valori correnti di x e y e gli incrementi relativi sono stati moltiplicati per 65536, e' possibile aggiungere gli incrementi ai valori originali ripetitivamente, per definire i nuovi valori di x e y, ma devono essere letti solo i bytes superiori di x e y, essendo la parte intera dei numeri.

Questo semplice ma efficace approccio puo' essere usato quando e' richiesto il risultato finale in forma intera. Se si vogliono cifre decimali, sono necessari processi piu' complessi. Lavorare in virgola fissa e' un concetto abbastanza lineare, ma ha delle complicazioni nei dettagli. Lo schema e' di adottare un numero fisso di decimali, diciamo due. In questo caso, tutti i numeri saranno moltiplicati per 100, come fattore di scala. Quando viene inserito un numero, sara' normalmente incluso il punto decimale, e il valore risultante di DPL sara' annotato ed usato per eseguire tutte le necessarie correzioni. Con un input di 23.0, d'altra parte, sarebbe richiesta una moltiplicazione per dieci.

I numeri modificati possono essere sommati o sottratti senza difficolta' poiche' hanno la stessa scala, ma le routines di moltiplica e divisione devono essere modificate. Dopo che due numeri sono stati moltiplicati, il risultato deve essere diviso

per 100, altrimenti quel moltiplicatore sarebbe applicato due volte. Prima di una divisione, il dividendo deve essere moltiplicato per 100, o il fattore di scala verrebbe perso.

L'output del risultato finale richiede solo che il punto decimale sia posto nella posizione corretta.

Si deve fare attenzione a possibili overflow. Una semplice moltiplicazione di 10 per 20 diventa una moltiplicazione di 1000 per 2000 e il risultato prima della correzione e' 2'000'000, corretto a 20'000. I numeri doppi seguiranno chiaramente la regola, e anche questi straripperanno con un valore non corretto di circa 2'000'000'000 dando un valore correto di 200'000.00. Se venisse usato un fattore di scala maggiore di 100, la limitazione del valore massimo aumenterebbe.

La soluzione si trova nelle parole triple, che richiedono un'intera nuova famiglia di manipolatori e di operatori.

Alternativamente, puo' essere considerata una forma di virgola mobile. Cio' sarebbe inevitabilmente molto piu' lento che non operarndo con numeri interi ma aprirebbe nuove possibilita'.

Un sistema a singola precisione puo' essere basato su un numero doppio con gli otto bits superiori utilizzati come esponente e i rimanenti 24 utilizzati come mantissa. Questo corrisponde al tipico sistema BASIC della precisione singola. La mantissa e' sempre "mormalizzata" cosi' che il suo bit piu' significativo e' sempre vero, essendo l'esponente decrementato quando la mantissa e' scalata (shifted) a sinistra e incrementato quando e' scalata a destra. Ci devono essere due bits per i segni, uno della mantissa e uno dell'esponente. Siccome il bit piu' significativo della mantissa e' sempre 1, vero, viene talvolta usato per contenere il bit segno della mantissa.

Per l'addizione e la sottrazione, e' necessario eguagliare gli esponenti dei due numeri in causa, incrementando l'esponente minore e decrementando l'esponente associato. Per la moltiplicazione gli esponenti vengono sommati e le mantisse moltiplicate. In un modo o nell'altro, la virgola mobile e' complessa e non ci si deve avvicinare ad essa a cuor leggero.

Coloro che hanno studiato il funzionamento interno del BASIC Spectrum possono essere in grado di usare la routine per virgola mobile contenuta in esso, ma l'approccio ad essa e' troppo complesso per essere studiato qui. E' utile tenere a mente, tuttavia, che la ROM BASIC contiene molte routines che potrebbero essere incorporate nel FORTH se venissero trovate le tecniche adatte.

In particolare, sarebbe bello poter accedere alle funzioni esponenziali e trigonometriche. E' possibile eseguire tali funzioni in FORTH, ma solo in modo relativamente crudo. La radice quadrata puo' essere calcolata tramite l'iterazione di:

$$S_{n+1} = 1/2(X/S_n + S_n)$$

Dove S_0 e' un valore arbitrario e S_n e' una approssimazione della radice quadrata di X. Quando S_n e' troppo grande, viene approssimativamente dimezzato ad ogni iterazione. Quando e' troppo piccolo viene aumentato. Approssimativamente, K+2

iterazioni produrranno generalmente un risultato ragionevolmente accurato per numeri fino a 2^k, se S_o viene posto uguale a 2^K/4.

Un punto interessante che ne deriva, e' che i calcoli trigonometrici possono talvolta essere rimpiazzati da una tecnica basata sui vettori. L'idea e' che le coordinate x e y possono essere combinate in un singolo vettore R dall'espressione:

$$R^2=SQR(a^2 + b^2)$$

L'angolo effettivo puo' essere espresso da a/b, e questo, con R, specifica completamente il vettore risultante. I vettori possono essere sommati o sottratti sommando e sottraendo i valori a e b, ed e' possibile in tal modo eseguire operazioni come definire una circonferenza- ma per fare cio', sono necessari calcoli in virgola mobile.

UN PROGRAMMA FINALE

Come ultima dimostrazione di alcune tecniche assortite del FORTH, ecco, per finire, un programma piuttosto vasto. Gioca a zero e croce[5] tridimensionale con considerevole successo, anche se puo' essere battuto, se sapete come fare e avete abbastanza concentrazione da tener d'occhio quello che fa il computer.

Sinceramente, e' il mio programma preferito avendolo scritto e riscritto per un'intera schiera di computers, includendone alcuni che erano stati progettati per compiti molto piu' seri. In codice macchina, le decisioni necessarie vengono prese in una frazione di secondo, ma in BASIC possono richiedere ben oltre un minuto. La versione FORTH richiede circa quindici secondi, il che e' appena accettabile, ed e' almeno cinque volte piu' veloce del BASIC.

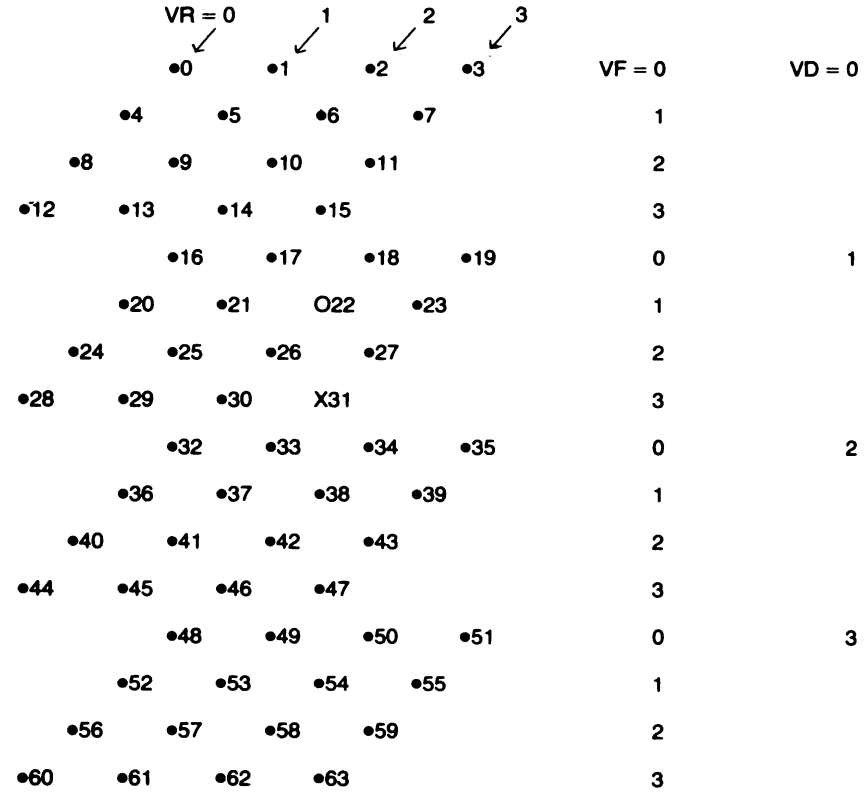
La ragione per cui e' richiesto tanto tempo risiede semplicemente nella vastita' dei calcoli implicati. In primo luogo, le dimensioni del programma vengono accorciate facendo a meno della solita tabella delle linee possibili nell'ambito della matrice cubica e al computer viene chiesto di calcolare queste linee da regole predefinite. Per ogni posizione data sono coinvolte sei o otto linee, e ci sono sessantaquattro posizioni. Devono essere controllate quattro posizioni per valutare lo stato di ogni linea.

Una volta che il contenuto della linea sia stato elaborato ed espresso in un codice numerico, deve essere valutato il livello di priorita' per la linea e aggiunto al totale perche' venga studiata la posizione; e questo deve essere ripetuto per ogni linea che passa attraverso quella posizione. Per finire deve essere trovata la priorita' maggiore.

Il diagramma del display aiutera' a spiegare alcune delle variabili. Vengono usati i numeri da 1 a 3 per le coordinate; alcuni giocatori potrebbero preferire i numeri da 1 a 4, nel qual caso bastera' una semplice sostituzione alla routine

[5] Gioco simile al nostro tris che si svolge su una scacchiera di 4x4 caselle. Nel caso specifico si gioca in tre dimensioni, dunque in un cubo ideale di 4x4x4 caselle. (N.d.T.)

INPROC. I numeri vengono inseriti nell'ordine VD, VF e VR. La posizione indicata verra' evidenziata sul video e vi sara' data la possibilita' di cambiare idea prima che l'inserimento venga fatto effettivamente.



OX3 Formato video e coordinate

CH e' il numero della posizione, usato principalmente per il riferimento all'array e per segnare l'ultimo inserimento; e' uguale a 16*VD+4*VF+VR.

Quindi vengono quattro flags , un contenitore temporaneo per la priorita', e una variabile OL che viene usata per trovare la priorita' maggiore.

Ci sono 4 arrays. AE e' un array a un byte che contiene l'inserimento di O e di X, AP e' un array a una parola che contiene le piorita', AX contiene le posizioni in una linea e AW e' l'indice delle priorita' e contiene il peso di ogni

configurazione di linea. Notate come viene definito AW, in modo simile ad un'istruzione BASIC DATA, ma con spazi da entrambi i lati delle virgole.

INIT cancella i flags e gli arrays e DISP rappresenta il display. POSCALC calcola i valori di VD ,VF e VR corrispondenti al valore di CH e viene usato da CURSOR, che evidenzia l'ultima posizione iserita come definito da CH, mettendo uno spazio inverso in modo OVER. Poiche' CURSOR segue sempre DISP questi sono combinati in PLAY.

Vengono stabilite due posizioni sul video da P1 e P2 e viene definito DROP2 per scartare il 20S.

La routine di input comincia con GETN , che attende il rilascio di un tasto, visualizza il carattere relativo e tenta di convertirlo in un numero in base 10. Il successo mette un flag vero in TOS, il numero in 20S e il codice inserito in 30S. Se l'input non e' numerico, TOS contiene un flag falso e 20S contiene il codice.

INPOS definisce la posizione del display P1 e chiama GETN . Se il flag che ritorna in TOS e' falso, il codice fornito da TOS viene controllato per vedere se e' 82 ("R"), che chiama un altra partita. In questo caso viene definito BE e TOS viene messo a 1 per uscire dalla routine a UNTIL. Se l'input e' numerico, il codice viene scartato da DROP2 e viene chiamato INPROC per ottenere il resto dell'input, controllandone la validita' ed offrendo la possibilita' di cambiare idea prima di inserire effettivamente uno 0 . Se vogliamo un limite di input di 1-4, invece di 0-3, la posizione calcolata in INPROC deve essere cambiata in:

```
ROT 1 - 16 * + 1 - SWAP 1 - 4 * + DUP AE + C@
```

INPOS ripete finche' TOS contiene 1 quando viene raggiunto UNTIL.

Poi viene la routine che calcola le priorita'. Viene chiamato CALC3 con l'array AX messo alla posizione di una data linea. Questo controlla il contenuto di ogni posizione a turno, aggiungendo 1 per uno "O", 5 per un "X". Cio' produce un numero caratteristico del formato di una linea. Per esempio, ad una linea che contiene OOX verrebbe assegnato il numero 7 . Guardando il settimo dato di AW notiamo che la priorita' e' zero, poiche' la linea e' "morta" contenendo sia "O" che "X". Una linea contenente OOOO, d'altro canto verrebbe rappresentata dal numero 4, e il quarto dato di AW e' ancora 0 perche' e' troppo tardi per preoccuparsi di priorita'; il gioco e' vinto. Se la linea contiene XXX, il suo numero e' 15 e questo da' una priorita' di 896, urge l'inserimento di un X per vincere.

Infatti, le priorita' sono predefinite per risparmiare tempo. Se il numero della linea e' 15, la posizione vuota viene localizzata e riempita, e il flag BX viene messo vero, cosicche' la vincita di X sia dichiarata immediatamente al ritorno. Analogamente, una linea con il numero 4 mette BO vero per dichiarare vincitore lo O.

In generale, la priorita' per la linea viene sommata a VP, che accumula le priorita' per tutte le linee che passano attraverso un particolare punto.

CALC2 calcola le posizioni in una linea usando i dati forniti da CALC1 . Ogni volta che viene chiamato CALC2, ci sono cinque numeri nello stack che esprimono la formula per una data linea. Per esempio, una linea verticale ha la formula:

```
4*VF + VR + 16*n
```

dove n va da 0 a 3.

Siccome le routines di calcolo sono complesse e proclivi all'errore nel definire il codice sorgente, e' utile definire una forma temporanea di CALC3 come segue:

```
: CALC3 CR 4 0 DO 12 * AX + ? LOOP ;
```

Questo dovrebbe essere definito dopo il vero CALC3 , ma prima di CALC1 . Questo visualizzera' una lista di quattro o sette gruppi di posizioni a formare linee per ogni valore di CH. Cio' consente che possa venir fatto un controllo sul sistema di selezione della linea.

Un punto importante che concerne queste routines e' l'uso di I' in CALC2 . Quando viene chiamata questa routine da CALC, viene messo un numero extra nel return stack ed e' necessario scavare fino a 2ORS per trovare il contatore del ciclo.

Le routines di calcolo vengono chiamate da CALCP che scandisce attraverso tutti i valori di CH da 0 a 63, controllando la priorita' contro ogni valore e inserendolo nell'array AP. La routine controlla anche se c'e' uno stato vero in BO e BX e, se uno di questi viene trovato, si esce dalla routine, saltando anche la successiva routine SELECT.

Siccome CALC3 mette una priorita' zero per ogni locazione occupata, potrebbe sembrare utile a risparmiar tempo saltare la chiamata a CALCP per tali locazioni. Questa sembra una buona idea, ma non lo e', perche' significherebbe che una linea 0000 non verrebbe mai scandita e la vittoria dell'O non verrebbe mai riconosciuta...

La routine SELECT scandisce l'array AP mettendo OL al valore letto se e' il piu' alto trovato sino ad allora e definendo CH per un controllo. Idealmente, la scansione dovrebbe iniziare in un punto a caso, e la casualita' viene simulata mettendo 56*VR nello stack, ma sarebbe meglio un vero numero casuale. Quando la scansione e' completa viene definito il numero identificato da CH. Se OL e' rimasto a zero, tutte le linee sono "morte" e BD viene messo a 1 per segnalare che la partita e' terminata in parita'.

Per finire, la routine a livello superiore OX3 deve unire tutte queste routines. Ci sono due cicli principali, uno per ogni gioco, che devono includere INIT e un ciclo interno per ogni turno di gioco. Dopo PLAY INPOS, viene fatto un controllo al flag di fine BE e, se e' vero, vengono saltate CALCP e SELECT. La struttura CASE viene usata per controllare i quattro flags che riportano il risultato e alterano i valori dello stack per determinare se deve essere preso il ciclo interno o esterno. Nessun programma FORTH e' mai perfetto, e questo non fa eccezioni. E' stato scelto per sottolineare un certo numero di punti speciali e potete divertirvi a tentare di migliorarlo. In questo caso visualizzatelo per esteso sullo schermo per fare facilmente i cambiamenti, ma non siate troppo generosi. Avete

150 linee in dieci schermi e, cosi' com'e' avete bisogno di circa 190 linee. Potreste usare due sets di schermi, ma questo puo' non essere conveniente.

Un programma di Zero e Croce tridimensionale

```
( OX3)
: TASK ;
0 VARIABLE VR                    (Coordinate da sinistra a destra)
0 VARIABLE VF                    (Coordinate da dietro a fronte)
0 VARIABLE VD                    (Coordinate dall'alto al basso)
0 VARIABLE CH                    (Numero della posizione)
0 VARIABLE BO                    (Flag O VINCE)
0 VARIABLE BD                    (Flag DISEGNA)
0 VARIABLE BE                    (Flag fine)
0 VARIABLE BX                    (Flag X VINCE)
0 VARIABLE VP                    (Contiene la priorita')
0 VARIABLE OL                    (Trova la priorita' piu' alta)
0 VARIABLE AE 62 ALLOT           (Array dello stato di gioco)
0 VARIABLE AP 126 ALLOT          (Array priorita')
0 VARIABLE AX 6 ALLOT            (Array definizione linea)
1 VARIABLE
AW 2 , 16 , 256 , 0 , 4 , 0 , 0 , 0 , 0 , 32 , 0 , 0 , 0 , 0 ,
   896 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,   (Basi di priorita')
: INIT AE 64 ERASE AP 128 ERASE
       0 BO ! 0 BD ! 0 BE ! 0 BX ! ;   (Cancella arrays e flags)
: DISP CLS 0 4 0 DO                  (Visualizza lo stato di gioco)
                 4 0 DO
                   CR 4 1 - 2 * SPACES
                   4 0 DO
                     DUP AE + C@ DUP
                     IF EMIT
                     ELSE ." ." DROP
                     ENDIF
                     SPACE SPACE
                   LOOP
                 LOOP CR
               LOOP ;
: POSCAL CH @ 16 /MOD VD ! 4 /MOD VF ! VR ! ;
: CURSOR POSCAL VD @ 5 * VF @ + 1+       (Posizione Verticale)
                VR @ 3 * VF @ 2 * - 8    (Posizione orizzontale)
                AT 1 GOVER 2 INK         (Posiziona il cursore)
                ."  " 0 GOVER 0 INK ;
: PLAY DISP CURSOR ;
: P1 9 18 AT ;                           (Posizione 1 del testo)
: P2 21 0 AT ;                           (posizione 2 del testo)
: DROP2 SWAP DROP ;
: GETN KEY DUP EMIT DUP 10 DIGIT ;       (Accetta una cifra)
: INPROC GETN
```

```
        IF DROP 2 GENTN          (Se numerica scarta il codice)
          IF DROP2               (Se numerica scarta il codice)
            ROT 16 * + SWAP
            4 * + DUP AE + C@ (Calcola la posizione, controlla)
          IF P2 ." Posizione presa       "
          ELSE CH ! PLAY P2        (Se e' libera, definisci CH,
             ." E' giusto?                              visualizza)
             KEY 89 - 0=    (Lascia 0 se inserimento scorretto)
          ENDIF DUP                  (Duplica il flag)
          IF 79 AE CH @ + C!          (Se e' 1 inserisci l'O)
             P2 20 SPACES           (e cancella il messaggio)
          ENDIF
        ELSE 0                        (Non numerico, lascia 0)
        ENDIF
      ELSE 0                          (Non numerico, lascia 0)
      ENDIF ;
: INPOS BEGIN
        P1 GETN
          IF DROP2 INPROC      (Se numerico, scarta il codice)
          ELSE 82 - 0=       (Altrimenti controlla se e'  "R")
            IF 1 BE ! 1      (Se e' "R",BE=1,TOS=1)
            ENDIF 1          (Se e' un'altra lettera, TOS=1)
          ENDIF
      UNTIL ;                (Ripeti se TOS=0)
: CALC3 0 4 0 DO
          1 2 * AX + @       (Leggi il numero della posizione)
          AE + C@            (Leggi i contenuti)
          79 - 0=            (Controlla se e' "O")
          IF SWAP 1+ SWAP    (Se e' "O" incrementa TOS)
          ENDIF
          88 - 0=            (Controlla se e' "X")
          IF 5 +             (Se e' "X" aggiungi 5)
          ENDIF
        LOOP                (Numero della linea chiave in TOS)
        AE CH @ + C@ 0=           (La posizione CH e' libera?)
        IF DUP 2 * AW + @ VP + !  (Se e' cosi' somma priorita'
        ENDIF                                          a VP)
        DUP 4 - 0=          (Controlla se la linea e' 0000)
        IF 1 BO !           (Se e' 0000 poni a 1 BO)
        ENDIF
        15 - 0=             (Controlla se la linea e' XXX)
        IF 8 0 DO         (Se e' XXX trova la posizione libera)
           I AX + @ AE + C@ 0=
           IF AX I + @ CH !    (Se e' libera poni a 1 CH)
              99 CH @ AE + C!  (E inserisci X)
           ENDIF
           2+ LOOP
           1 BX !              (Poni a 1 BX)
        ENDIF ;
: CALC2 VD @ * SWAP            (Calcola la posizione)
        VF @ * + SWAP
        VR @ * + SWOP
        I' * + +
```

```
        AX I' + 2 * + ! ;
: CALC1 POSCAL
        4 0 DO 0 1 0 4 16 CALC2 LOOP CALC3
        4 0 DO 0 4 1 0 16 CALC2 LOOP CALC3
        4 0 DO 0 16 1 4 0 CALC2 LOOP CALC3
        VF @ VR @ - 0=
        IF 4 0 DO 0 5 0 0 16 CALC2 LOOP CALC3
           VD @ VF @ - 0=
           IF 4 0 DO 0 21 0 0 0 CALC2 LOOP CALC3
           ENDIF
        ENDIF
        VD @ VR @ - 0=
        IF 4 0 DO 17 0 4 0 CALC2 LOOP CALC3
           VD @ VF @ + 3 - 0=
           IF 4 0 DO 12 13 0 0 0 CALC2 LOOP CALC3
           ENDIF
        ENDIF
        VD @ VF @ - 0=
        IF 4 0 DO 0 20 1 0 0 CALC2 LOOP CALC3
           VD @ VR @ + 3 - 0=
           IF 4 0 DO 3 19 0 0 0 CALC2 LOOP CALC3
           ENDIF
        ENDIF
        VD @ VR @ + 3 - 0=
        IF 4 0 DO 3 15 0 4 0 CALC2 LOOP CALC3
           VF @ VR @ - 0=
           IF 4 0 DO 15 11 0 0 0 CALC2 LOOP CALC3
           ENDIF
        ENDIF
        VF @ VR @ + 3 - 0=
        IF 4 0 DO 3 3 0 0 16 CALC2 LOOP CALC3
        ENDIF
        VD @ VF @ + 3 - 0=
        IF 4 0 DO 12 12 1 0 0 CALC2 LOOP CALC3
        ENDIF ;
: CALCP 1 0 CH ! 0 VP !                 (TOS=1,CH=0,VP=0)
        BEGIN
          CALC1 B0 @ 0=                 (Calcola la priorita')
          IF VP @ AP CH @ 2 * + !       (Se B0=0 mette prior. in AP)
             0 VP !                     (Azzera VP)
             BX @                       (Se BX=1 cambia il flag a 1)
             IF DROP 0 64               (e mette a fine il contatore)
             ELSE CH @ DUP 1 + CH ! (Incrementa CH,lascia copia)
             ENDIF
             64 - 0=                    (Se CH=64 lascia 1)
          ELSE DROP 0 1                 (Se B0=0 cambia lo stack)
          ENDIF                         (Se TOS=0 ripeti, altrimenti)
        UNTIL ;                         (esce col flag per SELECT)
: SELECT BEGIN WHILE                    (Lo salta se TOS=0)
           0 0L !                    (Azzera la "priorita' maggiore")
           VR @ 56 *                    (Numero quasi casuale)
           64 0 DO
            2+ 128 MOD                  (Avanza puntatore in stack)
```

```
            DUP AP + @ OL @ >       (Confronta AP(n) con OL)
            IF DUP AP + @ OL !      (Se AP(n) maggiore, OL=AP(n))
               DUP 2 / CH !         (E definisce CH dal TOS)
            ENDIF
          LOOP
          88 CH @ AE + C!     (Definisce la posizione "X" scelta)
          DROP 0          (Assicura che non ci siano ripetizioni)
        REPEAT
        OL @ 0=
        IF 1 BD !                   (Se OL=0 poni a 1 BD)
        ENDIF ;
: OX3 BEGIN INIT                (Punto di ricorsione del gioco)
        BEGIN PLAY INPOS        (Fa la mossa)
          BE @ 0=               (Controlla il flag fine)
          IF CALCP SELECT       (Se BE=0 trova la mossa X)
          ENDIF
          0 0 P2                (Condizione CASE=0)
        CASE BE @ 0= OF
             DROP 1 ENDOF       (Se BE=1, TOS=1)
             BD @ 0= 0F
             ." Gioco tracciato     "(Se BD=1 riporta il disegno)
             DROP 1 ENDOF            (e TOS=1)
             BO @ 0= OF
             ." 0 Vince             "(Se BO=1 riporta O Vince)
             DROP 1 ENDOF            (e TOS=1)
             BX @ 0= OF
             PLAY ." X Vince        "(Se BX=1 riporta con display
             DROP 1 ENDOF              e TOS=1)
        ENDCASE
      UNTIL
      P1 ." Un'altra partita?" KEY 89 -
    UNTIL ;
```

APPENDICE A:

Il Dizionario

I contenuti del dizionario del FORTH Abersoft sono definiti in forma condensata. Le parti in codice macchina sono espresse in pseudo-codici relativi alle funzioni del processore Z80. I campi parametri sono tradotti dagli indirizzi del campo codice ai nomi di parole. Una determinata parola puo' essere localizzata facendo riferimento alla lista associata nell'indice principale.

I registri dello Z80 vengono usati come segue:

BC contiene il puntatore delle istruzioni (IP), che da' l'indirizzo della prossima associazione da implementare.
DE viene usato per contenere i dati da mettere nello stack quando deve essere inserita (pushed) piu' di una parola (il registro W del FORTH). Contiene inoltre i valori incrementali e l'ampiezza dei salti.
HL viene usato per contenere i dati da mettere nello stack, il suo contenuto viene inserito dopo il contenuto di DE quando devono essere inserite due parole.
IX contiene il puntatore dell'area utente.
SP e' il puntatore dello stack.

Questi sono gli usi speciali dei registri. Tutti i registri possono essere utilizzati per altri scopi, se necessario.

Le definizioni del dizionario non sono facili da seguire poiche' ci sono molti rimandi, ma questo e' inevitabile. L'esame di alcune delle funzioni piu' complesse spesso suggerira' formati utili per nuove parole.

E' utile ricordare che una parola deve essere definita prima che possa essere usata in ulteriori definizioni, cosicche' i costituenti di una definizione si troveranno sempre prima della definizione stessa.

Il programma FORTH comincia con un certo numero di variabili e di costanti. Benche' queste non facciano strettamente parte del dizionario, sono necessarie per capire come operano alcune funzioni del dizionario.

5E06	S0	Contenuto iniziale del puntatore dello stack
5E08	R0	Contenuto iniz. del puntatore del return stack
5E0A	TIB	Indirizzo del Buffer di Input terminale
5E0C	WIDTH	Lunghezza massima della parola
5E0E	WARNING	Flag di controllo dei messaggi
5E10	FENCE	Limite di protezione del dizionario
5E12	DP	Puntatore del dizionario
5E14	VOC-LINK	Indirizzo d'inizio delle ricerche nel voc.
5E16	BLK	Numero del blocco
5E18	IN	Puntatore del buffer testo
5E1A	OUT	Puntatore dell'output
5E1C	SCR	Ultimo schermo usato
5E1E	OFFSET	Numero del set di schermi

```
5E20    CONTEXT     Indirizzo d'inizio del dizionario
5E22    CURRENT     Indirizzo d'inizio del dizionario
5E24    STATE       Modo del sistema
5E26    BASE        Base di rappresentazione numerica
5E28    DPL         Locazione del punto decimale
5E2A    FLD         Non usato
5E2C    CSP         Contenuto del puntatore dello stack
5E2E    R#          Editing del cursore, ecc.
5E30    HLD         Indirizzo dell'ultimo carattere in output
5E32
5E34
5E36
5E38
5E3A
5E3C
5E3E
5E40    00
5E41    C3 B0 6D    Salto a COLD a 6DB0
5E44    00
5E45    C3 93 6D    Salto a WARM a 6D93
5E48    01 01
5E4A    00 0E
5E4C    49 81
5E4E    0C 00
5E50    66 5E
5E52    CB40        Valore iniziale per 5E06
5E54    CBE0                            5E08
5E56    CB40                            5E0A
5E58    001F                            5E0C
5E5A    0000                            5E0E
5E5C    8159                            5E10
5E5E    8159                            5E12
5E60    77AC                            5E14
5E62
5E64
5E66    CE00        Sorgente per IX
5E68                Puntatore del return stack
```

Comincia ora il dizionario vero e proprio. Per permettere possibili modifiche, i punti d'ingresso sono definiti tramite numeri d'associazione anziche' con indirizzi, ma gli indirizzi effettivi possono essere determinati abbastanza semplicemente con l'aiuto di -FIND. Le parole segnate con * vengono eseguite anche in modo compilazione.

```
1 PUSHDE            Mette (push) il contenuto di DE nello stack
2 PUSHHL            Mette il contenuto di HL nello stack
3 NEXT1             HL=BC.BC=BC+2
4 NEXT2             HL=(HL).vai a (HL)
```

Dove, come sopra, le definizioni sono ravvicinate, esse vengono eseguite una dopo l'altra. PUSHDE e PUSHHL sono usati di ritorno da altri moduli per inserire dati nello stack del calcolatore e quindi il puntatore interpretativo IP (contenuto in BC) viene usato per vedere la successiva

associazione alla quale la routine salta. IP viene incrementato per puntare all'associazione per la prossima azione. Questo e' il controllo centrale dell'intero sistema FORTH.

5 LIT HL=(BC).BC=BC+2.Vai a PUSHHL

La parola memorizzata nelle due locazioni che seguono l'associazione di LIT, viene letta in HL e messa nello stack. LIT, in effetti, identifica la parola come dato, non come associazione.

6 EXECUTE POP HL.Vai a NEXT2

Il TOS viene messo in HL (POP) come associazione alla routine da eseguirsi.

7 BRANCH HL=BC.DE=(HL).HL=HL+DE.BC=HL.Vai a NEXT1

La parola memorizzata nella coppia di locazioni che seguono BRANCH viene sommata a IP, l'interpretazione viene ripresa all'indirizzo risultante.

8 0BRANCH POP HL.Se HL=0 Vai a BRANCH.
Altrimenti BC=BC+2.Vai a NEXT1

Questo e' il salto condizionale eseguito solo se TOS=0. Altrimenti il salto viene evitato incrementando IP.

9 (LOOP) DE=1

10 HL=(RSP).(HL)=(HL)+DE.DE=(HL)
HL=HL+2.Se D era negativo, vai a 11.
Altrimenti metti il flag segno a DE-(HL).
Vai a 12

11 Metti il flag segno a (HL)-DE.

12 Se negativo vai a BRANCH
Altrimenti HL=HL+2.(RSP)=HL
BC=BC+2.Vai a NEXT1.

Questa e' la forma compilata di LOOP. HL legge il puntatore del Return Stack e DE viene sommato a TORS. Viene quindi comparato 2ORS con TORS tenendo conto del segno di DE (vedere (+LOOP) piu' sotto). Se l'indice in TORS non ha passato ne' raggiunto il valore limite in 2ORS, viene chiamato BRANCH per saltare a un punto immediatamente successivo a (DO), l'ampiezza del salto necessario e' stata calcolata durante la compilazione. Altrimenti il RSP viene messo indietro di due parole, scartando, in effetti, le due parole relative. L'IP salta il dato che rappresenta l'ampiezza del salto e NEXT1 continua l'azione.

13 (+LOOP) POP DE.Vai a 10

Invece di essere messo a 1, come per (LOOP), DE e' nello al valore di TOS, segue (LOOP).

14 (DO) (RSP)=(RSP)-4.POP DE.HL=(RSP)
(HL)=DE.POP DE.HL=HL+2
(HL)=DE.Vai a NEXT1.

Questa e' la forma compilata di (DO). Vengono fatti due inserimenti nel Return Stack. Il nuovo TORS e' preso da TOS (valore iniziale dell'indice) e il nuovo 2ORS e' preso da 2OS (valore limite dell'indice).

15 I HL=(RSP).DE=(HL).PUSH DE.Vai a NEXT1

Il contenuto di TORS viene copiato in TOS. All'interno di

un ciclo DO questo trasferisce il valore corrente dell'indice.

```
16  DIGIT       POP HL.POP DE.A=E-48
                Se A e' negativo vai a 18.
                Altrimenti se A e' minore di 10 vai a 17.
                A=A-7.
                Se A e' minore di 10 vai a 18.
17              Se A e' maggiore di L vai a 18.
                E=A.HL=1.Vai a PUSHDE.
18              L=H. Vai a PUSHHL.
```

La base numerica da TOS viene messa in HL. Il codice ASCII in 2OS viene messo in DE. Il codice e' convertito in un valore numerico. Se tale valore e' negativo, maggiore della base numerica o in altro modo non valido, viene raggiunto 18. Viene posto uno zero come TOS (si assume che la base numerica non ecceda 255, cosi' H=0). Altrimenti il numero e' posto in 2OS, con TOS=1 ad indicare il successo della conversione.

```
20  (FIND)      POP DE.
                POP HL.PUSH HL.
                Se (DE) XOR (HL) AND 3FH <> 0 vai a 26.
21              INC HL.INC DE.
                Se (DE) XOR (HL) <> 0 vai a 25.
                Se il bit 7 di (DE) XOR (HL) = 0 vai a 21.
                HL=DE+5
22              Scambia (SP) e HL
23              DEC DE.
24              A=(DE). Se il bit 7 di A = 0 vai a 23
                E=A.D=0.HL=1. Vai a PUSHDE.
25              Se (DE) XOR (HL) = 1 vai a 27.
26              INC DE.A=(DE). Se il bit 7 di A = 0 vai a 26
27              INC DE. Scambia DE e HL.DE=(HL)
                Se DE <> 0 vai a 20.
                Altrimenti POP HL.HL=0. vai a PUSHHL.
```

Questa e' una routine che cerca una corrispondenza con il testo puntato da 2OS, la routine parte a TOS. Il testo di riferimento e' il nome di una parola e la routine lo ricerca attraverso i campi nome del dizionario. Il puntatore di ricerca viene messo (POP) in DE, e il puntatore del testo di riferimento viene messo in HL e quindi rimesso nello stack. Se i codici (bytes lunghezza) ai quali puntano sono diversi rispetto ai bits 0-5, la routine salta a 26 (notate che il bit "macchia" (SMUDGE) non viene considerato). Altrimenti, i puntatori vengono incrementati. La routine ricorre a 21 finche'; o viene trovata una diversita' o viene trovato il bit 7 di (DE), che segna la fine del nome del dizionario. In quest'ultimo caso HL=DE+5, cosicche' HL punta all'indirizzo del campo parametri che viene messo nello stack al posto del puntatore del testo di riferimento. Viene quindi ripetitivamente decrementato DE finche' viene trovato il byte lunghezza della parola del dizionario, il byte lunghezza viene quindi messo nello stack seguito da un

flag vero. Se viene riscontrata la non corrispondenza di un carattere, viene raggiunto 25. Se il bit 7 del nome del dizionario era vero, la routine salta a 27. Altrimenti, e se una non corrispondenza del byte lunghezza porta la routine a 26, DE viene ripetitivamente incrementato finche' (DE) ha il bit 7 vero. Un ulteriore incremento fa puntare a DE l'indirizzo del campo associazione. Viene letta l'associazione e se non e' zero la routine salta a 20. se l'associazione e' zero, e' stata raggiunta la fine del dizionario, e il TOS e'sostituito con un flag zero (falso) a indicare che non e' stata trovata corrispondenza.

```
28  ENCLOSE     POP DE.POP HL.PUSH HL.
                A=E.D=A.E=FFH.DEC HL.
29              INC HL.INC E. Se (HL)=A vai a 29
                D=0.PUSH DE.D=A.A=(HL)
                Se A<>0 vai a 30.
                Altrimenti D=0.INC E.PUSH DE.DEC E.PUSH DE.
                Vai a NEXT1
30              A=D.INC HL.INC E.Se A=(HL) vai a 31.
                Se (HL)<>0 vai a 30.
                D=0.PUSH DE.PUSH DE.Vai a NEXT1.
31              D=0.PUSH DE.INC E.PUSH DE.Vai a NEXT1.
```

Questa routine scandisce il testo per localizzare i delimitatori. Il delimitatore viene prelevato da TOS e messo in DE, l'indirizzo d'inizio per la scansione viene copiato da 20S in HL. Il codice del delimitatore viene copiato in A e D. E viene messo a -1 e HL decrementato. Un ciclo in 29 cerca quindi il primo carattere non delimitatore, avanzando HL ed E. Quando si esce dal ciclo, D=0 e DE viene messo nello stack (PUSH) (mettendo il primo carattere non delimitatore in 30S). Se (HL)=0, E+1 viene messo nello stack (mettendo il prossimo delimitatore in 20S) e poi E (mettendo in TOS il primo carattere non incluso).
Se (HL)<>0, un ciclo in 30 scandisce il testo finche' viene trovato un delimitatore (va a 31) oppure (HL)=0. Nel primo caso i valori messi nello stack sono E ed E+1, nel secondo caso due volte E.

```
32  EMIT        Viene chiamato il codice macchina a 274
                OUT +! OUT viene incrementato.
33  KEY         Viene chiamato il codice macchina a 266.
```

Per dettagli, vedere le routines in codice macchina.

```
34  ?TERMINAL   HL=0. Viene chiamato il C.M. a 263.
35  CR          Viene chiamato il codice macchina a 276.
36  CMOVE       HL=BC.POP BC.POP DE. Scambio fra (SP) e HL
                Se BC=0 vai a 37
                Altrimenti LDIR.POP BC.
37              Vai a NEXT1
```

LDIR esegue l'azione di copiatura, ma e' necessario che BC contenga il parametro lunghezza, cosi' l'IP passa a HL,quindi allo stack, e poi ancora a BC quando e' stato eseguito LDIR.

```
38    U*          POP DE.POP HL.PUSH BC.
                  B=H.A=L.CALL 39
                  PUSH HL.H=A.A=B.B=H.CALL 39.
                  POP DE.C=D.HL=HL+BC.PUSH DE.
                  Vai a PUSHHL.
39                HL=0.C=8
40                HL=2*HL.RL A.Se non c'e' il carry vai a 41.
                  HL=HL+DE.A=A+carry
41                DEC C. Se c<>0 vai a 40.
                  RETURN
```

La routine a 39 moltiplica A per DE, con il risultato in HL. E' in primo luogo utilizzato per moltiplicare il byte inferiore di 2OS per TOS, quindi per moltiplicare il byte superiore di 2OS per TOS. I due risultati vengono combinati in un numero doppio, la meta' inferiore messa nello stack da DE, la meta' superiore da HL. Notate che anche qui l'IP viene salvato nello stack per rendere disponibile BC per altri usi.

```
42    U/MOD       HL=4+SP.E=(HL).(HL)=C
                  INC HL.D=(HL).(HL)=B.POP BC.POP HL.
                  Se BC e' maggiore di HL vai a 43.
                  Altrimenti HL=FFFFH.DE=FFFFH. Vai a 48.
43                A=16
44                HL=2*HL RL A. Scambia DE e HL.
                  HL=2*HL.Se non c'e' carry vai a 45.
                  INC DE.AND A.
45                Scambia DE e HL.RR A.PUSH AF.
                  Se non c'e' carry vai a 46.
                  A=A AND L.HL=HL-BC-carry
                  Vai a 47.
46                HL=HL-BC.Se non c'e' carry vai a 47.
                  HL=HL+BC.DEC DE
47                INC DE.POP AF.DEC A.Se A<>0 vai a 44.
48                POP BC.PUSH HL.PUSH DE.Vai a NEXT1
```

Quando viene messo (PUSH) un dato nello stack Z80, il puntatore dello stack punta all'ultimo byte inserito. Aggiungendo 4 a SP, questi dunque puntera' al byte alto di 3OS. Questi viene messo in DE e sostituito da IP preso da BC. Viene prelevato TOS e messo in BC e 2OS in HL. La routine divide 2OS/3OS per TOS. Se TOS non e' maggiore di 2OS, viene messo nello stack FFFFH due volte. Altrimenti viene eseguita una routine di divisione iterativa su una base ristabilita, che lascia un resto vero. Questo viene posto nello stack da HL, il risultato da DE.

```
49    AND         POP DE.POP HL.HL=HL AND DE.
                  Vai a PUSHHL.
50    OR          POP DE.POP HL.HL=HL OR DE.
                  Vai a PUSHHL.
51    XOR         POP DE.POP HL.HL=HL XOR DE.
                  Vai a PUSHHL.
52    SP@         HL=SP.Vai a PUSHHL.
53    SP!         DE=(IX+6).Scambia DE,HL.RSP=HL.
                  Vai a NEXT1
```

IX punta a 5E40 (vedi COLD) e viene usata per accedere all'area variabili.

```
54  RP@         HL=(RSP).Vai a PUSHHL.
55  RP!         DE=(IX+8).Scambia DE,HL.(RSP)=HL.
                Vai a NEXT1
56  ;S          HL=(RSP).BC=(HL).HL=HL+2.
                (RSP)=HL.Vai a NEXT1.
```

Questa funzione chiave termina l'interpretazione di una definizione tramite due punti o di uno schermo. L'IP viene ripreso dal Return Stack.

```
57  LEAVE       HL=(RSP).DE=(HL).HL=HL+2.
                (HL)=DE.Vai a NEXT1
```

Usato all'interno di un ciclo DO LOOP, LEAVE copia il valore dell'indice corrente al posto del valore limite cosi' che si uscira' dal ciclo al prossimo LOOP.

```
58  >R          POP DE.HL=(RSP).(HL)=DE.HL=HL-2
                (RSP)=HL.Vai a NEXT1.
59  R>          HL=(RSP).DE=(HL).HL=HL+2
                (RSP)=HL.PUSH DE.Vai a NEXT1.
```

Queste due trasferiscono dati fra TOS e TORS, aggiustando gli stacks. La prossima funzione copia TORS in TOS senza cambiare TORS.

```
60  R           Vai a 15
```

R e' identica a I nell'azione.

```
61  0=          POP HL.Se HL=0 allora HL=1, altrimenti HL=0.
                Vai aPUSH HL.
62  0<          POP HL.HL=2*HL. Se non c'e' carry HL=0,
                altrimenti HL=1. Vai a PUSHHL.
63  +           POP DE.POP HL.HL=HL+DE.Vai a PUSHHL.
64  D+          HL=SP+6.DE=(HL).(HL)=BC.
                POP BC.POP HL.HL=HL+DE.
                Scambia DE,HL.POP HL.
                HL=HL+BC+carry.
                POP BC.PUSH DE. Vai a PUSHHL.
```

L'IP viene salvato nello stack in 40S. Vengono usate due diverse addizioni per sommare i due doppi numeri.

```
65  MINUS       POP DE.HL=0-DE.Vai a PUSHHL.
66  DMINuS      POP HL.POP DE.DE=0-DE.
                HL=0-HL-carry.PUSH DE.Vai a PUSHHL.
67  OVER        POP DE.POP HL.PUSH HL.Vai a PUSHDE.
68  DROP        POP HL.Vai a NEXT1
69  SWAP        POP HL.Ecambia (SP) e HL. Vai a PUSHHL.
70  DUP         POP HL.PUSH HL.Vai a PUSHHL.
71  2DUP        POP HL.POP DE.PUSH DE.PUSH HL.Vai a PUSHDE.
72  +!          POP HL.POP DE.(HL)=(HL)+E.INC HL.
                (HL)=(HL)+D+carry.Vai a NEXT1.
73  TOGGLE      POP DE.POP HL.(HL)=XOR E.Vai a NEXT1
74  @           POP HL.DE=(HL).PUSH DE.Vai a NEXT1.
75  C@          POP HL.L=(HL).H=0.Vai a PUSHHL.
76  2@          POP HL.HL=HL+2.DE=(HL).PUSH DE.HL=HL-2.
                DE=(HL).PUSH DE.Vai a NEXT1.
77  !           POP HL.POP DE.(HL)=DE.Vai a NEXT1.
78  C!          POP HL.POP DE.(HL)=E.Vai a NEXT1.
```

```
79    2!            POP HL.POP DE.(HL)=DE.POP DE.HL=HL+2.
                    (HL)=DE.Vai a NEXT1.
*80                 ?EXEC         Errore se non modo esecuzione.
                    !CSP          Salva il puntatore stack in CSP
                    CURRENT @
                    CONTEXT !     CONTEXT=CURRENT
                    CREATE        Crea un titolo nel dizionario.
                    ]             Riprende la compilazione
                    (;CODE)       Punta al codice seguente (81).
81                                HL=(RSP).(HL)=BC.HL=HL-2.
                                  (RSP)=HL.INC DE.BC=DE.
```

Questa e' un'altra routine chiave. L'esecuzione dei due punti esegue i controlli visti piu' sopra, quindi definisce un'associazione a 81. L'IP interpreta poi le parole che seguono, definendo associazioni o parole di dati per rappresentarli. Questo forma il campo parametri della nuova parola. Il processo termina con il punto e virgola:

```
*82 ;               ?CSP          Errore se SP<>CSP.
                    COMPILE       Compila l'associaz. nel diz.
                    ;S
                    SMUDGE        Ridefinisce il bit "macchia"
                    [             Sospende la compilazione.
```

Notate che due punti e punto e virgola, si incontrano solo in modo esecuzione, i due punti inseriscono il modo compilazione e il punto e virgola ristabilisce il modo esecuzione.

```
83  NOOP                          NOOP non fa niente, tranne
                                  andare a NEXT1.
84  CONSTANT        CREATE        Inserisce una parola nel diz.
                    SMUDGE        TOGGLE del bit "macchia"
                    ,             (Virgola) Mette TOS nel diz.
                    (;CODE)       Punta al codice seguente.
85                                INC DE.Scambia DE,HL.DE=(HL).
                                  PUSH DE.vai a NEXT1.
```

CONSTANT crea una parola nel dizionario che fa riferimento al codice a 85. Questo preleva la parola seguente e la mette in TOS.

```
86  VARIABLE        CONSTANT      Crea una costante.
                    (;CODE)       Punta al codice seguente.
87                                INC DE.PUSH DE.Vai a NEXT1.
```

Il codice a 87 mette nello stack l'indirizzo della variabile.

```
88  USER            CONSTANT      Crea una costante.
                    (;CODE)       Punta al codice seguente.
89                                HL=IX+DE.Vai a PUSHHL.
```

Si fa riferimento ad una variabile nello spazio generale di lavoro, usando IX e uno spiazzamento a un singolo byte prelevato da E. L'indirizzo della variabile va in TOS.

```
90  0                             Costante 0 in TOS
91  1                             Costante 1 in TOS
92  2                             Costante 2 in TOS
93  3                             Costante 3 in TOS
94  BL                            Costante 20H (Codice spazio) in
```

```
                                  TOS
95   C/L                          Costante 40H (Caratteri per
                                  linea) in TOS
96   FIRST                        costante CBE0H (Inizio del
                                  primo buffer) in TOS
97   LIMIT                        Costante D000H (Fine
                                  dell'ultimo buffer) in TOS
98   B/BUF                        Costante 80H (Bytes per buffer)
                                  in TOS
99   B/SCR                        Costante 8 (Blocchi per
                                  schermo) in TOS
     Le parole di cui sopra usano il codice a 85.
100  +ORIGIN      LIT 5E40 +      Somma 5E40 (origine nominale) a
                                  TOS
101  SO                           Indirizzo 5E06 in TOS
102  RO                           Indirizzo 5E08 in TOS
103  TIB                          Indirizzo 5E0A in TOS
105  WIDTH                        Indirizzo 5E0C in TOS
106  WARNING                      Indirizzo 5E0E in TOS
107  FENCE                        Indirizzo 5E10 in TOS
108  DP                           Indirizzo 5E12 in TOS
109  VOC-LINC                     Indirizzo 5E14 in TOS
110  BLK                          Indirizzo 5E16 in TOS
111  IN                           Indirizzo 5E18 in TOS
112  OUT                          Indirizzo 5E1A in TOS
113  SCR                          Indirizzo 5E1C in TOS
114  OFFSET                       Indirizzo 5E1E in TOS
115  CONTEXT                      Indirizzo 5E20 in TOS
116  CURRENT                      Indirizzo 5E22 in TOS
117  STATE                        Indirizzo 5E24 in TOS
118  BASE                         Indirizzo 5E26 in TOS
119  DPL                          Indirizzo 5E28 in TOS
120  FLD                          Indirizzo 5E2A in TOS
121  CSP                          Indirizzo 5E2C in TOS
122  R#                           Indirizzo 5E2E in TOS
123  HLD                          Indirizzo 5E30 in TOS
     Queste parole usano il codice a 89.
124  1+           1 +             Somma 1 a TOS
125  2+           2 +             Somma 2 a TOS
126  HERE         DP @            Mette in TOS il puntatore del
                                  dizionario.
127  ALLOT        DP +!           Somma TOS al puntatore del
                                  dizionario.
128  ,            HERE 2 !        Memorizza TOS a HERE, somma 2 a
                  ALLOT           DP (virgola).
129  C,           HERE C!         Memorizza il byte di TOS a
                  1 ALLOT         HERE, DP=DP+2.
130  -                            POP DE.POP HL.HL=HL-DE.
                                  Vai a PUSHHL.
131  =            - 0=            Se TOS=20S, TOS=1 altrimenti 0.
132  <                            POP DE.POP HL.Se il bit 7 di
                                  D XOR H = 0 allora HL=HL-DE (i
                                  segni differiscono) se H e'
```

```
                                positivo allora HL=0 altrimenti
                                1.Vai a PUSHHL.
133  U<          2DUP XOR 0<    Se i segni di TOS e 2OS
                                differiscono mette un flag
                                vero.
                 0BRANCH
                 000C           Se il flag e' falso vai a 134.
                 DROP           Scarta TOS.
                 0< 0=          Mette un flag vero se il nuovo
                                TOS e' positivo.
                 BRANCH 0006    Fine.
134              - 0<           Mette un flag vero se TOS
                                eccede 2OS.
135  >           SWAP <         Scambia TOS e 2OS ed esegue la
                                funzione inversa.
136  ROT                        POP DE.POP HL.Scambia (SP),HL.
137  SPACE       BL EMIT        Scrive uno spazio.
138  -DUP        DUP            Duplica TOS
                 0BRANCH
                 0004           Fine se TOS=0
                 DUP            Duplica TOS
139  TRAVERSE    SWAP           Stack: indirizzo n (n= + o - 1)
140              OVER +         stack: indirizzo n + n
                 LIT 007F
                 OVER           Stack: indirizzo n + n 7FH
                                indirizzo + n
                 C@ <           Confronta (indirizzo + n) con
                                7FH.
                 0BRANCH
                 FFF0           Se (indirizzo + n) non e'
                                maggiore di 7FH vai a 140.
                 SWAP DROP      Altrimenti scarta n, lasciando
                                indirizzo + n.
     Questo esegue una scansione attraverso un campo nome del
     dizionario nella direzione determinata dal segno di n,
     finche' non viene trovato un byte col bit 7 vero.
141  LATEST      CURRENT @ @    TOS=(CURRENT)
142  LFA         LIT 0004 -     Sottrae 4 da TOS.
143  CFA         2 -            Sottrae 2 da TOS.
144  NFA         LIT 0005 -     Sottrae 5 da TOS.
                 LIT FFFF       TOS=-1.
                 TRAVERSE       Esegue una scansione indietro
                                attraverso il campo nome.
145  PFA         1 TRAVERSE     Esegue una scansione in avanti
                                attraverso il campo nome.
                 LIT 0005 +     Aggiunge 5 a TOS.
146  !CSP        SP @           Legge il puntatore dello stack.
                 CSP !          Lo scrive in CSP.
147  ?ERROR      SWAP           Stack: flag n
                 0BRANCH
                 0008           Se il flag e' vero vai a 148.
                 ERROR          Riporta l'errore
                 BRANCH 0004    Fine
```

```
148                 DROP            Scarta il numero dell'errore.
149   ?COMP         STATE @ 0=      Flag vero se STATE=0
                    LIT 0011        Errore numero 17
                    ?ERROR          Riporta l'errore se il flag e'
                                    vero.
150   ?EXEC         STATE @         Flag falso se STATE=0
                    LIT 0012        Errore numero 18
                    ?ERROR          Riporta l'errore se il flag e'
                                    vero.
151   ?PAIRS        -               Flag falso se TOS=2OS
                    LIT 0013        Errore numero 19
                    ?ERROR          Riporta errore se il flag e'
                                    vero.
152   ?CSP          SP @ CSP @ -    Confronta SP con CSP.
                                    Flag falso se e' uguale.
                    LIT 0016        Errore numero 20
                    ?ERROR          Riporta l'errore se il flag e'
                                    vero.
153   ?LOADING      BLK @ 0=        Flag vero in TOS se si sta
                                    usando TIB.
                    LIT 0016        Errore numero 22
                    ?ERROR          Riporta l'errore se il flag e'
                                    vero.
154   COMPILE       ?COMP           Errore se non si e' in modo
                                    compilazione
                    R>              TORS in TOS
                    DUP 2+          Stack: TORS TORS+2
                    >R              TOS in TORS
                    @ ,             Memorizza (TORS) in HERE
*155  [             0 STATE !       Mette a zero STATE
156   ]             LIT 00C0
                    STATE !         Mette 129 in STATE
157   SMUDGE        LATEST
                    LIT 0020        Esegue TOGGLE sul bit 7 di
                    TOGGLE          LATEST
158   HEX           LIT 0010
                    BASE !          Mette BASE=16
159   DECIMAL       LIT 000A
                    BASE !          Mette BASE=10
160   (;CODE)       R>              TORS in TOS
                    LATEST
                    PFA CFA !       Scrive TORS al campo codici di
                                    LATEST
*161  ;CODE         ?CSP            Controlla che SP=CSP
                    COMPILE
                    (;CODE)         Compila (;CODE)
                    [               Mette STATE=0
                    SMUDGE          Esegue TOGGLE sul bit 7 di
                                    LATEST
162   <BUILDS       0 CONSTANT      Stabilita una costante
163   DOES>         R>              TORS in TOS
                    LATEST PFA !    Copia nel campo parametri di
                                    LATEST
```

```
                (;CODE)       Associazione al codice seguente
164                           HL=(RSP)-2.(HL)=BC.(RSP)=HL.
                              INC DE.Scambia DE,HL.BC=(HL).
                              Vai a PUSHHL.
    L'IP viene messo nel Return Stack e re-inizializzato da
    (DE+1), mentre DE+1 va in TOS.
165 COUNT         DUP 1+      Stack: ind ind+1
                  SWAP C@     Stack: ind+1 (ind)
    Lo stack contiene inizialmente l'indirizzo del byte
    lunghezza di una stringa di testo. L'indirizzo dell'inizio
    del testo viene messo in 2OS con il byte lunghezza in TOS.
166 TYPE          -DUP        Duplica TOS se non e' 0
                  0BRANCH
                  0018        BRANCH a 168 se TOS=0
                  OVER +      Stack: ind ind+lung
                  SWAP        Stack: ind+lung ind
                  (DO)
167               I C@        Legge (I) in TOS
                  EMIT        Scrive TOS
                  (LOOP) FFFB Ciclo a 167
                  BRANCH 0004 Fine
168               DROP        Scarta l'indirizzo
169 -TRAILING     DUP 0       Stach: ind lung lung 0
                  (DO)
170               OVER OVER   Stack: ind lung ind lung
                  + 1 -       Stack: ind lung (ind+lung-1)
                  C@ BL -     confronta (ind+lung-1) con il
                              codice spazio
                  0BRANCH
                  0008        Vai a 171 se viene riscontrata
                              corrispondenza.
                  LEAVE       Definisce la condizione di
                              uscita dal ciclo.
                  BRANCH 0006 Vai a 172
171               1 -         Decrementa la lunghezza
172               (LOOP) FFE0 Ciclo a 170.
    Un testo stringa viene scandito all'indietro, il byte
    lunghezza viene decrementato ad ogni codice spazio che si
    incontra, finche' non si trova un codice diverso dallo
    spazio.
173 (.")          R           TORS copiato in TOS
                  COUNT       Aggiusta l'indirizzo e la lung.
                  DUP 1+      Stack: ind lung lung+1
                  R> + >R     TORS=TORS+lung+1
                  TYPE        Emette la stringa
    Viene aggiornato TORS e viene emessa la stringa definita.
*174 ."           LIT 0022    Codice per " (delimitatore)
                  STATE @     TOS=STATE
                  0BRANCH     Se siamo in modo esecuzione vai
                  0014        a 175.
                  COMPILE (.")
                  WORD        Memorizza il testo a HERE
                  HERE C@     Legge il byte lunghezza
```

```
                    1+ ALLOT        Riserva lung+1 bytes
                    BRANCH 000A     Fine
175                 WORD            Memorizza il testo a HERE
                    HERE
                    COUNT TYPE      Emette il testo
```

Questa parola agisce in maniera piuttosto diversa in modo esecuzione e in modo compilazione. In compilazione definisce (.") con il testo. In esecuzione visualizza il testo.

```
176 EXPECT          OVER + OVER     Stack: ind ind+limit ind
                    (DO)
                    KEY DUP         Stack: ind input input
                    LIT 000E
                    + ORIGIN @      TOS=(5E4E)
                    =               Flag zero se TOS<>input
                    0BRANCH
                    002A            Se il flag e' zero vai a 179
                    DROP DUP I =    Flag zero se I<>ind
                    DUP             Duplica il flag
                    R> 2 - + >R     TORS=TORS-2+flag
                    0BRANCH
                    000A            Se il flag e' zero vai a 178
                    NOOP NOOP       Ritardo?
178                 BRANCH 0008     Codice per il cursore a sinist.
                    BRANCH 0028     Vai a 182
179                 DUP             Stack: ind input input
                    LIT 000D        Codice newline
                    =               Flag zero se input<>newline
                    0BRANCH
                    000E            Se il flag e' zero vai a 180
                    LEAVE           Termina il ciclo
                    DROP BL 0       Stack: ind codice-spazio zero
                    BRANCH 0004     Vai a 181
180                 DUP             Stack: ind input input
181                 I C!            Mette input in (I)
                    0 I 1+ !        Azzera la prossima locazione
182                 EMIT            Scrive il carattere in TOS
183                 (LOOP) FF9C     Ciclo a 177
                    DROP            Cancella lo stack, fine.
184 QUERY           TIB @           Indirizzo del buffer di input
                                    terminale in TOS
                    LIT 0050        Lunghezza limite (80 bytes)
                    EXPECT          Vedi sopra
                    0 IN !          Azzera IN.
*185 (Vedi la nota) BLK @           Numero del blocco in TOS
                    0BRANCH
                    002A            Vai a 187 se e' in uso TIB
                    1 BLK +!        Incrementa il numero del blocco
                    0 IN !          Azzera IN
                    BLK @           Numero del blocco in TOS
                    B/SCR 1 -       Blocchi per schermo -1 in TOS
                    AND 0=          Flag zero se (TOS AND 2OS)=0
                    0BRANCH
```

```
                    0008            Se il flag e' zero vai a 186
                    ?EXEC '         Errore se non si e' in modo
                                    esecuzione
                    R> DROP         Scarta TORS
186                 BRANCH 0006     Fine
187                 R> DROP         Scarta TORS
```

Il campo nome per questa parola contiene solo CIH,80H che indica il codice 0. Esso abilita i blocchi con un codice nullo.

```
188 FILL                            HL=BC.POP DE.POP BC.
                                    Scambia (SP),HL
                                    Scambia DE,HL
189                                 Se BC=0 vai a 190
                                    (DE)=L.INC DE.DEC BC.
                                    Vai a 189
190                                 POP BC.Vai a NEXT1
```

DE, quindi HL, contengono il carattere da usarsi. Il numero di caratteri da inserire e' contenuto in BC,mentre HL, poi DE, contiene l'indirizzo d'inizio, che e' incrementato dopo ogni inserimento.

```
191 ERASE           0 FILL          Vedi sopra.
192 BLANKS          BL FILL         Vedi sopra.
193 HOLD            LIT -1 HLD +! Decrementa HlD
                    HLD @ C!        Memorizza TOS in (HLD)
194 PAD             HERE
                    LIT 0044 +      TOS=HERE+68
```

Il buffer PAD non e' ad un indirizzo fisso ma si trova sopra il dizionario ad una distanza di 68 bytes.

```
195 WORD            BLK @           TOS=numero del blocco
                    0BRANCH
                    000C            Se e' in uso TIB vai a 196
                    BLK @           TOS=numero del blocco
                    BLOCK           TOS=indirizzo del blocco
                    BRANCH 0006     Vai a 197
196                 TIB @           TOS=Indirizzo del TIB
197                 IN @ +          Somma IN
                    SWAP            Stack: ind delimitatore
                    ENCLOSE         Stack: ind offset1 offset2
                                    offset3
                    HERE LIT 0022
                    BLANKS          Inserisce 34 codici spazio
                    IN +!           Somma offset3 a IN
                    OVER            Stack: ind offset1 offset2
                                    offset1
                    - >R
                    R HERE C!       TORS=offset2-offset1
                                    (HERE)=TORS
                    + HERE 1+ R>    Stack: ind+offset1
                                    HERE+1 TORS
                    CMOVE           Copia TORS bytes da ind+offset1
                                    a HERE+1
198 (NUMBER)        1+ DUP          Stack: X/Y ind+1 ind+1
                    >R              TORS=ind+1
```

```
                    C@              TOS=(ind+1)
                                    (Codice numerico ASCII)
                    BASE @          TOS=BASE
                    DIGIT           Converte il codice ASCII in un
                                    numero.
                    0BRANCH
                    002C            Se non e' valido vai a 200
                    SWAP BASE @     Stack: X numero Y BASE
                    U*              Stack: X numero Y*BASE
                    DROP            Scarta la parola superiore del
                                    numero doppio prodotto
                    ROT BASE
                    @ U*            Stack: numero Y*BASE X*BASE
                    D+              Stack: X*BASE+(Y*BASE+numero)
                    DPL @ 1+        TOS=DPL+1
                    0BRANCH
                    0008            Se DPL=-1 vai a 199
                    1 DPL +!        Incrementa DPL (per contare i
                                    caratteri dopo il punto)
199                 R>              Riprendi l'indirizzo da TORS
                    BRANCH FFC6     Vai a 198
200                 R>              Cancella TORS
    Questa routine viene chiamata normalmente da NUMBER, che
    mette a zero X/Y. Il processo merita un esame minuzioso.
201 NUMBER          0 0 ROT DUP     Stack: 0 0 ind ind
                    1+ C@           Stack: 0 0 ind (ind+1)
                    LIT 002D        Codice per il segno meno
                    =               Flag zero se (ind+1) <> codice
                                    segno meno.
                    DUP             Duplica il flag
                    >R              Flag in TORS
                    +               Stack: 0 0 ind+flag (un passo
                                    dopo il codice segno)
                    LIT -1
202                 DPL !           Definisce DPL
                    (NUMBER)        Vedere sopra
                    DUP C@          Stack: numero/numero ind (ind)
                    BL -            Lo confronta con il codice
                                    spazio
                    0BRANCH
                    0016            Se e' codice spazio vai a 203
                    DUP C@          Stack: numero/numero ind (ind)
                    LIT 002E -      lo confronta con il codice del
                                    punto decimale
                    0 ?ERROR        Errore se non e' il punto
                                    decimale
                    0               Mette a 0 DPL e' il punto
                                    decimale.
                    BRANCH FFDC     Vai a 202
203                 DROP R>         Riprende il flag segno da TORS
                    0BRANCH
                    0004            Se il flag=0 fine.
                    DMINUS          Nega il risultato.
```

NUMBER produce sempre un doppio numero come risultato. Il byte superiore viene scartato da INTERPRET se non c'e' il punto decimale.

```
204  -FIND          BL              Codice spazio in TOS
                    WORD            Copia il nome a HERE
                    HERE            Definisce il puntatore di
                                    riferimento
                    CONTEXT @ @     Definisce il puntatore di
                                    ricerca
                    (FIND)          Ricerca la parola
                    DUP             Stack: PFA lung flag flag
                    0=              Inverte il flag
                    0BRANCH
                    000A            Fine se viene trovata
                                    corrispondenza
                    DROP            Scarta il flag
                    HERE            Definisce il puntatore di
                                    riferimento
                    LATEST          Mette il puntatore di ricerca a
                                    LATEST
                    (FIND)          Ricerca ancora
```

Se (FIND) fallisce, viene lasciato nello stack solo il flag, cosicche' non v'e' bisogno di scartare il byte lunghezza e PFA. Viene poi fatta una ricerca a partire da LATEST anziche' da (CONTEXT).

```
205  (ABORT)        ABORT
```

Questa parola permette all'utente usi speciali di (ABORT)

```
206  ERROR          WARNING
                    @ 0<            Flag vero se WARNING negativo
                    0BRANCH
                    0004            Se WARNING non e' negativo vai
                                    a 207
                    (ABORT)
207                 HERE
                    COUNT TYPE      Scrive la parola errata
                    (.") "?"        Scrive il punto di domanda
                    MESSAGE         Scrive il numero d'errore o il
                                    testo
                    SP!             Resetta SP
                    BLK @ -DUP      Numero del blocco, duplicato se
                                    non e' zero
                    0BRANCH
                    0008            Se e' in uso TIB vai a 208
                    IN @ SWAP       Stack: blocco IN
208                 QUIT            Restituisce il controllo
                                    all'utente.
209  ID.            PAD LIT 0020
                    LIT 005F FILL Riempie il buffer PAD con 32
                                    codici 5F
                    DUP PFA LFA     Stack: NFA LFA
                    0VER -          Stack: NFA LFA-NFA
                    PAD SWAP        Stack: NFA PAD LFA-NFA
                    CMOVE           Copia LFA-NFA bytes da NFA a
```

```
                        PAD
                PAD COUNT       Controlla la lunghezza della
                                copia
                LIT 001F AND    Limita a 31 bytes
                2DUP            Stack: ind lung ind lung
                + 1 - DUP       Stack: ind lung ind+lung-1
                @               TOS=codice dell'ultima lettera
                LIT FF7F AND    Rimuove il bit 7
                SWAP !          Ripristina i bytes modificati
                TYPE SPACE      Scrive la parola, aggiunge uno
                                spazio.
210  CREATE     -FIND           Ricerca il nome della parola
                0BRANCH
                0010            Vai a 211 se e' unico
                DROP            Scarta la lunghezza
                NFA ID.         Scrive il nome della parola
                LIT 0004        Messaggio 4
                MESSAGE         Scrive il numero o il messaggio
                SPACE           Aggiungi uno spazio
211             HERE DUP C@     Stack: HERE (HERE)
                WHIDTH @ MIN    Lunghezza limite 31 caratteri
                1+ ALLOT        ALLOT per uno in piu'
                DUP             Stack: HERE HERE
                LIT 00A0
                TOGGLE          TOGGLE dei bit 5 e 7 del byte
                                lunghezza
                HERE 1 -        HERE modificato da ALLOT
                LIT 0080
                TOGGLE          TOGGLE del bit 7 dell'ultima
                                lettera
                LATEST ,        Definisce il campo associazioni
                CURRENT @ !     Viene letto current come un
                                indirizzo. (CURRENT)=HERE
                HERE 2+ ,       Definisce il campo codici
*212 [COMPILE]  -FIND           Ricerca il nome della parola e
                                lo inserisce a HERE
                0=              TOS=0 se trovato
                0 ?ERROR        Errore 0 se non trovato
                DROP            Scarta il byte lunghezza
                CFA ,           Definisce il campo codici
*213 LITERAL    STATE @
                0BRANCH
                0008            Se il modo e' esecuzione fine.
                COMPILE LIT ,   Definisce l'inserimento di LIT
*214 DLITERAL   STATE @
                0BRANCH
                0008            Se il modo e' esecuzione fine.
                SWAP
                LITERAL
                LITERAL
215  ?STACK     SP@ S0 @        Stack: SP S0
                SWAP            Stack: S0 SP
                U<              Flag vero se SP eccede S0
```

```
                    1 ?ERROR        Errore 1 se il flag e' falso
                    SP@ HERE
                    LIT 0080        Stack: SP HERE+128
                    U<              Flag vero se HERE+128 e'
                                    maggiore
                    LIT 0007
                    ?ERROR          Errore 7 se il flag e' vero
216 INTERPRET       -FIND           Ricerca il nome della parola e
                                    la inserisce a HERE
                    0BRANCH
                    001E            Se non viene trovato vai a 219
                    STATE @ <       Flag vero se la lunghezza e'
                                    minore di STATE
                    0BRANCH
                    000A            Se e' falso vai a 217
                    CFA ,           Inserisce CFA a HERE
                    BRANCH 0006     Vai a 218
217                 CFA EXECUTE     Esegue le funzioni indicate
218                 ?STACK          Controlla i limiti dello stack
                    BRANCH 001C     Vai a 222
219                 HERE
                    NUMBER          Interpreta la parola come un
                                    numero
                    DPL @ 1+        TOS=DPL+1
                    0BRANCH
                    0008            Se DPL=-1 vai a 220
                    DLITERAL        Definisce un numero doppio
                    BRANCH 0006     Vai a 221
220                 DROP            Scarta il byte superiore del
                                    numero doppio
                    LITERAL         Definisce un numero singolo
221                 ?STACK          Controlla i limiti dello stack
222                 BRANCH FFC2     Vai a 216
```

Il confronto del byte lunghezza con lo stato implementa le caratteristiche speciali di alcune parole. Notate che se il nome di una parola e' anche un numero valido nella BASE corrente, non sara' possibile inserire quel numero, perche' sara' sempre interpretato prima come parola.

```
223 IMMEDIATE       LATEST
                    LIT 0040
                    TOGGLE          TOGGLE del bit 6 di LATEST
224 VOCABULARY      <BUILDS         Stabilisce l'ingresso di una
                                    costante
                    LIT A081 ,      Memorizza A081 in HERE
                    CURRENT @
                    CFA ,           Memorizza CFA di CURRENT in
                                    HERE
                    HERE
                    VOC-LINC @ ,    Memorizza VOC-LINK in HERE
                    VOC-LINC !      Memorizza il precedente HERE in
                                    VOC-LINK
                    DOES>
225                 2 + CONTEXT !   Mette CONTEXT a CURRENT + 2
```

```
*226 FORTH                          Esegue il codice a 164
                                    Esegue 225
227    (spazio)                     Azzera il campo codici
228  DEFINITIONS     CONTEXT @
                     CURRENT !      CURRENT=CONTEXT
*229 (               LIT 0029       Il delimitatore e' la chiusa
                                    parentesi
                     WORD           Definisce un commento ma lo
                                    ignora
230  QUIT            0 BLK !        Definisce il blocco 0 (in uso
                                    TIB)
                     [              STATE=0
231                  RP!            Inizializza RSP
                     CR QUERY       Newline. Accetta un testo in
                                    input
                     INTERPRET      Interpreta il testo
                     STATE @ 0=     Flag vero se il modo e' esecuz.
                     0BRANCH
                     0007           Se il flag e' falso vai a 232
                     ." "ok"        Visualizza "ok"
232                  BRANCH FFE7    Vai a 231
233  ABORT           SP!            Inizializza SP
                     DECIMAL        Seleziona la rappresentazione
                                    decimale
                     ?STACK         Controlla i limiti dello stack
                     CLS CR         Cancella lo schermo, newline
                     .CPU           Scrive "48K Spectrum"
                     (.") testo     Scrive "fig-FORTH 1.1A"
                     CR             Newline
                     (.") testo     Scrive "(C) Abersoft: 1983"
                     CR             Newline
                     FORTH          Seleziona il vocabolario FORTH
                     DEFINITIONS    CURRENT=CONTEXT
                     QUIT           Ritorna il controllo all'utente
234  Ingresso da     WARM inizio    BC=indirizzo dell'associazione
                                    a 236
                                    IX=(5E66)=5E00
                                    HL=(5E52)=CB40
                                    SP=HL
                                    Vai a NEXT1 (e di la' a 236)
235                                 Associazione a CFA per WARM
236  WARM            EMPTY-
                     BUFFERS        Segna i buffers come vuoti
                     ABORT          Vedi sopra
237  Ingresso da     COLD inizio    (FLAGS2)=8 (Variabile BASIC)
                                    (hold)=0 (Vedi 266)
                                    BC=indirizzo dell'associazione
                                    a 267
                                    IX=(5E66)=5E00
                                    HL=(5E52)=CB40
                                    SP=HL
                                    Vai a NEXT1
238                                 Associazione a CFA per COLD
```

```
239  COLD          EMPTY-
                   BUFFERS       Segna i buffers come vuoti
                   LIT CBE0 USE ! Definisce il prossimo
                                  buffer=CBE0
                   LIT CBE0
                   PREV !        Definisce l'ultimo buffer=CBE0
                   DR0           OFFSET=0
                   LIT 5E52
                   LIT 5E66 @
                   LIT 0006 +
                   LIT 0010      Stack: 5E52 (5E66)+6 16
                   CMOVE         Copia 16 bytes, a partire da
                                  5E52 fino a 5E06
                   LIT 5E4C @    TOS=8149
                   LIT 6CF8 !    (6CF8)=8149 (campo associazioni
                                  di 227)
                   ABORT
240  S->D                       POP DE.HL=0.A=D AND 80H
                                Se A<>0 DEC HL.
                                Vai a PUSHDE
241  +-            0<           Se TOS e' negativo flag vero
                   0BRANCH
                   0004         Se il flag e' falso allora fine
                   MINUS        Nega TOS
242  D+-           0<           Flag vero se TOS e' negativo
                   0BRANC
                   0004         Se il flag e' falso allora fine
                   DMINUS       Nega la parola doppia
243  ABS           DUP +-       Se il TOS e' negativo negalo
244  DABS          DUP D+-      Se il TOS e' negativo nega la
                                parola doppia
245  MIN           2DUP >       Se 20S e' maggiore di TOS flag
                                vero
                   0BRANCH
                   0004         Se il flag e' falso vai a 246
                   SWAP
246                DROP         Scarta il maggiore
247  MAX           2DUP <       Se 20S e' minore di TOS flag
                                vero
                   0BRANCH
                   0004         Se il flag e' falso vai a 248
                   SWAP
248                DROP
249  M*            2DUP XOR >R  TORS=TOS XOR 20S
                                (Stack conservato)
                   ABS SWAP
                   ABS          Rende positivi TOS e 20S
                   U*           Prodotto di TOS*20S
                   R> D+-       Nega il risultato se il segno
                                di TOS 20S differiva
250  M/            OVER >R >R   Stack: X Y TORS=Z 20RS=Y
                   DABS         Converte a positivo il doppio
                                numero
```

```
                    R ABS           Converte a positivo Z
                    U/MOD           Stack: res quoz
                    R> R XOR        Negativo se il segno di XY e  Z
                                    differisce
                    +-              Se negativo nega il quoziente
                    SWAP
                    R> +-           Nega il resto se Z e' negativo
                    SWAP
251  *              M* DROP         Scarta la meta'  superiore  del
                                    prodotto
252  /MOD           >R S->D R>      Estende il segno di 20S
                    M/              Come sopra
253  /              /MOD SWAP
                    DROP            Scarta il resto
254  MOD            /MOD DROP       Scarta il quoziente
255  */MOD          >R              TOS in TORS
                    M*              Prodotto
                    R>              Riprende il TOS originale
                    M/              Resto Quoziente
256  */             */MOD SWAP
                    DROP            Scarta il resto
257  M/MOD          >R 0 R          Rende doppio 20S
                    U/MOD           Resto quoziente1
                    R> SWAP >R      Resto divisore
                                    TORS contiene quoziente1
                    U/MOD R>        Resto quoziente2 quoziente1
258  (LINE)         R>              Stack: linea   TORS: schermo
                    LIT 0040
                    B/BUF           Bytes per buffer
                    */MOD           Linea*64/bytes per buffer
                                    (Res e quoz)
                    R> B/SCR *      Res quoz schermo*B/SCR
                    +               Res quot+schermo*B/SCR
                    BLOCK +         Somma l'indirizzo del blocco
                    LIT 0040
259  .LINE          (LINE)          Vedi sopra
                    -TRAILING       Scarta gli spazi superflui
                    TYPE            Output
260  MESSAGE        WARNING @
                    0BRANCH 001E Se WARNING=0 vai a 262
                    -DUP            Duplica se non e' zero
                    0BRANCH 0014 Se messaggio 0 vai a 261
                    LIT 0004        Schermo 4
                    OFFSET @        Blocco contrapposto
                    B/SCR /         Divide per blocchi per schermo
                    -               Sottrae dal numero di schermi
                    .LINE           Scrive il messaggio
                    SPACE           Aggiunge uno spazio
261                 BRANCH 000D     Fine
262                 (.") testo      Scrive "MSG #"
                    .               Scrive il numero
```

Il codice seguente viene usato da ?TERMINAL (vedi 34), vengono coinvolte routines BASIC.

263 PUSH BC.PUSH DE.CALL 1F54.
HL=0
Se c'e' carry vai a 265
264 Se (5C08)=7 allora INC HL.
265 POP DE.POP BC.Vai a PUSHHl.

Il codice seguente viene usato da KEY (vedi 33). Vengono coinvolte routines BASIC.

266 PUSH BC.A=2.CALL 1601.
A=12.RST 10.A=1.RST 10.
267 (5C08)=0
268 A=(hold).RST 10.A=B.RST 10.
Se (5C08)=0 vai a 268
Se (5C08)<>6 vai a 271
HL=5C6A.(HL)=(HL) XOR 8
269 HL=hold.Se il bit 3 di A=1 vai a 270
(HL)=4CH.Vai a 267
270 (HL)=43H.Vai a 267
271 Se A<>0FH vai a 273
A=2.HL=5C41.
(HL)=(HL) XOR A
Se (HL)=0 vai a 272
A=(5C6A).Vai a 269
272 A=47H.(hold)=A.Vai a 267

In BASIC, 5C08 e' LASTK, 5C41 e' MODE, 5C6A e' FLAGS2. La routine da 273 in poi e' concepita principalmente per registrare certi tasti.

273 C6 (AND) diventa 5B [
C5 (OR) diventa 5D]
E2 (STOP) diventa 7E ~
C3 (NOT) diventa 7C |
CD (STEP) diventa 5C \
CC (TO) diventa 7B {
CB (THEN) diventa 7D }
Se A e' maggiore di A5H (dopo la conversione di cui sopra) vai a 267 (cioe' ignora l'input) altrimenti L=A.
H=0.A=12.RST 10.A=0.RST 10.A=20
RST 10.A=8.RST 10.POP BC.
Vai a PUSHHL.

Il codice seguente viene usato da EMIT (vedi 32). Di nuovo, vengono coinvolte routines BASIC.

274 PUSH BC.PUSH HL.A=2.
CALL 1601.POP HL.PUSH HL.
A=1.RST 10.A=(hold2).
Se A=0 vai a 275
CALL 1601.POP HL.PUSH HL.
A=L.RST 10
275 POP HL.A=FFH.(5C8C)=A.
POP BC.Vai a NEXT1

5C8C in BASIC e' SCRCT (Contatore di Scroll). Metterlo a FFH assicura lo scroll continuo. Il contenuto di hold2

```
     controlla la stampante (vedi 337).
     Il codice seguente e' usato da CR (vedi 35).
276                                  PUSG BC.A=2.CALL 1601.A=2DH.
                                     RST 10.A=(hold2).
                                     Se A=0 vai a 277.
                                     CALL 1601.A=0D.RST 10.
277                                  POP BC.(5C8C)=FFH.
                                     Vai a NEXT1.
278  USE                             TOS = (Blocco buffer   piu'
                                     vecchio)
279  PREV                            TOS = (Ultimo blocco buffer)
280  #BUFF                           TOS = (Numero di buffers disco)
281  +BUFF           LIT 0084 +      Stack: Ind+132
                     DUP LIMIT =     Stack: Ind+132 flag vero se
                                     Ind+132=LIMIT
                     0BRANCH 0006    Se il flag e' falso vai a 282
                     DROP FIRST      Sostituisce FIRST
282                  DUP PREV @ -    Lo confronta con PREV.
                                     (Lascia ind flag)
283  UPDATE          PREV @ @        TOS=contenuto di PREV
                     LIT 8000 OR
                     PREV @ !        Mette a 1 il bit 15   del
                                     contenuto di PREV.
284  EMPTY-BUFFERS   FIRST LIMIT
                     OVER -          Stack: FIRST LIMIT-FIRST
                     ERASE           Cancella l'area buffer con zero
                     LIMIT FIRST
                     (DO)            Da FIRST a LIMIT-1;
285                  LIT 7FFF I !    Mette 7FFF in (I)
                     LIT 0084        STEP 132
                     (+LOOP) FFF2    Ciclo a 285
286  DRO             0 OFFSET !      Mette a 0 OFFSET
287  BUFFER          USE @ DUP >R    Stack: (USE)   TORS: (USE)
288                  +BUF            Avanza al prossimo buffer
                     0BRANCH FFFC    Se PREV vai a 288
                     USE !           Segnalo come in uso
                     R @ 0<          Se l'ultimo  buffer  aggiornato
                                     (negativo) metti un flag vero
                     0BRANCH 0014    Se il flag e' falso vai a 289
                     R 2+ R @        (USE)=(USE)+2. ((USE))
                     LIT 7FFF AND    Azzera msb
                     0 R/W           Scrivi il buffer su disco
289                  R !             Definisce (USE) come scritto.
                     R PREV !        Mette PREV=USE
                     R 2+            Lascia (USE)+2
290  BLOCK           OFFSET @ +      Stack:n.blocco+OFFSET
                     >R              Trasferimento a TORS
                     PREV @ DUP      Stack: PREV PREV
                     @               Stack: PREV (PREV)
                     R - DUP         Stack: PREV (PREV)-TORS
                     +               Stack: PREV+(PREV)-TORS
291                  0BRANCH 0034    Se 0 vai a 293
                     +BUF 0=         Muove al prossimo buffer.  Vero
```

```
                                se PREV
                 0BRANCH 0014   Vai a 292 se falso
                 DROP           Scarta l'indirizzo del buffer
                 R BUFFER       Vedi sopra (n.blocco + OFFSET
                                fornisce i dati)
                 DUP            Duplica l'indirizzo
                 R 1 R/W        Legge nell'ultimo buffer
                 2 -            Modifica l'indirizzo
292              DUP @ R -      Confronta con TORS i dati del
                                buffer
                 DUP +          Duplica il TOS
                 0=             Inverte lo stato del flag
                 0BRANCH FFD6   Se il flag e' falso vai a 291
                 DUP PREV !     Definisce PREV
293              R> DROP        Scarta il TOS
                 2 +            Avanza l'indirizzo
294  LO                         TOS=D000 (Indirizzo del disco
                                RAM
295  HI                         TOS=FBFF (Fine del disco RAM)
296  R/W         >R             TORS=flag direzione
                                (0=scrittura,1=lettura)
                 B/BUFF *       Stack: ind blocco*128
                 LO +           Stack: ind blocco*128+D000
                 DUP HI >       Flag vero se eccede i limiti
                 LIT 0006       Errore 6
                 ?ERROR         Riporta l'errore se il flag e'
                                falso
                 R>             Cancella TORS, flag direzione
                                in TOS
                 0BRANCH 0004   Vai a 297 se e' falso
                 SWAP
297              B/BUF
                 CMOVE          Copia un buffer pieno come da
                                istruzioni
298  FLUSH       #BUFF
                 1+
                 0 (DO)
299              BUFFER DROP
                 (LOOP) FFF8    Ciclo a 299
300  LOAD        DUP 0=         Flag vero se c'e'
                                corrispondenza
                 LIT 0009       Errore numero 9
                 ?ERROR         Riporta l'errore se e' vero
                                (LOAD dello schermo 0)
                 BLK @          Legge il numero del blocco
                 >R             Numero blocco in TORS
                 IN @ >R        IN in TORS
                 0 IN !         IN=0
                 B/SCR *        schermo*B/SCR
                 BLK !          Scrive in BLK
                 INTERPRET
                 R> IN !        Riporta IN al valore originale
                 R> BLK !       Riporta BLK al valore originale
```

```
*301 -->            ?LOADING      Errore se non si e' in LOAD
                    0 IN !        IN=0
                    B/SCR BLK @   B/SCR BLK
                    OVER          B/SCR BLK BSCR
                    MOD           BLK resto di BLK/B/SCR
                    - BLK +!      BLK=BLK+BLK-resto
*302 ' (apostrofo)  -FIND 0=      Cerca il nome, inverte il flag
                    0 ?ERROR      Errore 0 se non viene trovato
                    DROP          Scarta il byte lunghezza
                    LITERAL
303  BACK           HERE - ,      Memorizza ind-HERE
*304 BEGIN          ?COMP         Errore se il modo non e'
                                  compilazione
                    HERE 1        Identifica il punto di
                                  ricorsione, mette a uno il
                                  riferimento a PAIRS
*305 ENDIF          ?COMP         Errore se il modo non e'
                                  compilazione
                    2 ?PAIRS      Errore se il riferimento a
                                  PAIRS non e' 2
                    HERE OVER -   Stack: a HERE-a
                    SWAP !        Scrive l'ampiezza della
                                  associazione
*306 THEN           ENDIF         Le due parole hanno lo stesso
                                  significato
*307 DO             COMPILE (DO)
                    HERE 3        Identifica il punto di
                                  ricorsione, riferimento a PAIRS
                                  uguale a 3
308 LOOP            3 ?PAIRS      Errore se il riferimento a
                                  PAIRS non e' = 3
                    COMPILE (LOOP)
                    BACK          Calcola l'ampiezza della
                                  associazione all'indietro
*309 +LOOP          3 ?PAIRS      Errore se il riferimento a
                    COMPILE (+LOOP)  PAIRS non e' 3
                    BACK          Calcola l'ampiezza della
                                  associazione all'indietro
*310 UNTIL          1 ?PAIRS      Errore se il riferimento a
                                  PAIRS non e' 1
                    COMPILE 0 BRANCH
                    BACK          Calcola l'ampiezza della
                                  associazione all'indietro
*311 END            UNTIL         Le due parole hanno lo stesso
                                  significato
*312 AGAIN          1 ?PAIRS      Errore se il riferimento a
                                  PAIRS non e' 1
                    COMPILE BRANCH
                    BACK          Calcola l'ampiezza della
                                  associazione all'indietro
*313 REPEAT         >R >R         TOS, 2OS al Return Stack
                    AGAIN
                    R> R>         TOS, 2OS al loro valore
```

```
                        iniziale
                2 -     Converte il riferimento a PAIRS
                        da 4 a 2 (vedi WHILE)
                ENDIF
*314 IF         COMPILE 0BRANCH
                HERE
                0 ,     Riserva spazio
                2       Riferimento PAIRS=2
*315 ELSE       2 ?PAIRS  Errore se il riferimento a
                        PAIRS non e' uguale a 2
                COMPILE BRANCH
                HERE
                0 ,
                SWAP
                2 ENDIF 2  Riferimento a PAIRS=2
*316 WHILE      IF 2+   Esegue IF, poi incrementa di 2
                        il riferimento a PAIRS
317 SPACES      0 MAX   Assicura un valore positivo
                -DUP    Duplica se non e' zero
                0BRANCH 000C Fine se TOS=0
                0 (DO)
318             SPACE   Emette uno spazio
                (LOOP) FFFC  Loop a 318
319  <#         PAD HLD !  Mette HLD=PAD
                DROP DROP  Scarta il bilanciamento dei
                        numeri
320  #>         HLD @   Legge HLD
                PAD OVER -  Stack: HLD PAD-HLD
321  SIGN       ROT     Porta il segno in TOS
                0<      Vero se TOS minore di zero
                0BRANCH 0008 Se TOS=0 allora fine
                LIT 002D  Codice del segno meno
                HOLD    Memorizza ad HLD
322  #          BASE @  Stack: X Y BASE
                M/MOD   Stack: resto doppio-quoziente
                ROT     Stack: doppio-quoziente resto
                LIT 0009 OVER Stack: doppio-quoziente resto
                        9 resto
                <       Vero se il resto eccede 9
                0BRANCH 0008 Vai a 323 se e' falso
                LIT 0007 +  Somma 7 al resto
323             LIT 0030 +  Somma 48 per formare il codice
                        ASCII
                HOLD    Memorizzalo in HLD
324  #S         #       Vedi sopra
                OVER OVER  Stack: numero-doppio
                        numero-doppio
                OR 0=   Flag vero se il numero e' =0
                0BRANCH FFF4 Se e' falso vai a 324
325  D.R        >R SWAP OVER Stack: Y X Y (XY numero doppio)
                        TORS=n
                DABS    Rende positivo XY
                <# #S   Mette il numero in PAD
```

```
                    SIGN #>         Aggiunge il segno
                    R> OVER -       Stack: ind cont n-cont
                    SPACES          Emette n-cont spazi
                    TYPE            Scrive il numero
```

Notate che il numero viene ridotto a zero da #S, cosi' il segno deve essere determinato dalla copia di Y.

```
326 .R              >R              n in TORS
                    S-->D           Segno esteso per formare un
                                    numero doppio
                    R>              Riprende n
                    D.R             Vedi sopra
327 D.              0               Chiama un campo di lunghezza 0
                    D.R             Vedi sopra
                    SPACE           Aggiunge uno spazio
328                 S-->D           Estende il segno per formare un
                                    numero doppio
                    D.              Vedi sopra
329 ?               @ .             Scrive il contenuto
                                    dell'indirizzo specificato in
                                    TOS
330 U.              0               Forma un numero doppio senza
                                    segno
                    D.              Vedi sopra
331 VLIST           LIT 0080
                    OUT !           OUT=128
                    CONTEXT @ @     Contenuto di CONTEXT in TOS
332                 OUT @
                    LIT 001F
                    LIT 0008        Stack: (CONTEXT) OUT 31 8
                    - >             Flag vero se OUT eccede 23
                    0BRANCH 000A    Se il flag e' falso vai a 333
                    CR 0 OUT        Newline. OUT=0
333                 DUP             Duplica il puntatore
                    ID.             Visualizza il nome
                    PFA LFA @       Legge il campo associazione
                    DUP 0=          Flag vero se e' zero
                    ?TERMINAL       Flag vero se BREAK
                    OR              O flag vero effettivo
                    0BRANCH FFD0    Se il flag e' falso ciclo a 332
                    DROP            Scarta l'ultimo indirizzo
334 LIST            DECIMAL
                    CR              Newline
                    DUP SCR !       Definisce lo schermo scelto
                    (.") testo      Scrive "SCR #"
                    .               Visualizza il numero di schermo
                    LIT 0010
                    0 (DO)
335                 CR              Newline
                    I               Numero di linea
                    LIT 0003 .R     Scrive in un campo di tre spazi
                    SPACE           Aggiunge uno spazio
                    I SCR @
                    .LINE           Visualizza la linea memorizzata
```

```
                    ?TERMINAL     Vero se BREAK
                    0BRANCH 0004 Se falso vai a 336
                    LEAVE
336                 (LOOP) FFE2   Vai a 335
                    CR            Newline
337 LINK                          POP HL.A=L.Se A<>0 allora A=3
                                  (hold2)=A.Vai a NEXT1
338 CLS                           PUSH BC.A=2.CALL 1601.
                                  CALL 0D6B.A=2.CALL 1601.
                                  POP BC.Vai a NEXT1
339 .CPU            (.") testo    Scrive "48K SPECTRUM"
    A questo punto viene riservata un'area per gli headers  del
    nastro della forma:
    03 44 49 53 43 20 20 20 20 20 20 FF 2B 00 D0 20 20
       D  I  S  C
    03 44 49 53 43 20 20 20 20 20 20 FF 2B 00 D0 20 20
       D  I  S  C
340 (TAPE)                        POP HL.PUSH BC.PUSH IX.
                                  A=L.HL=D000.IX=(inizio
                                  dell'area header).A=(5C72).
                                  CALL 075A.POP IX.POP BC.
                                  Vai a NEXT1
342 TEXT            HERE
                    C/L 1+        Stack: HERE C/L+1
                    BLANKS        Definisce una linea di spazi  a
                                  HERE
                    WORD          Definisce la stringa a HERE
                    HERE PAD C/L
                    1+ CMOVE      Copia la linea completa da PAD
343 LINE            DUP           Stack: linea linea
                    LIT FFF0 AND Limite a 15
                    LIT 0017      Errore numero 23
                    ?ERROR        Errore se non e' 0
                    SCR @         Legge lo schermo corrente
                    (LINE)        Definisce  l'indirizzo    della
                                  linea e lo considera
                    DROP          Scarta il conto della lunghezza
344 LOADT           1 (TAPE)
345 SAVET           FLUSH 0 (TAPE)
346 VERIFY          2 (TAPE)
348 2SWAP           ROT >R        Stack da a b c d ad a c d
                                  b in TORS
                    ROT R>        Stack: c d a b
349 SIZE            HERE
                    0 + ORIGIN -  TOS=HERE-ORIGIN
350 FREE            SP@ HERE -    TOS=SP-HERE
351 FORGET          CURRENT @
                    CONTEXT @ -   TOS=CURRENT-CONTEXT
                    LIT 0018      Errore numero 24
                    ?ERROR        Riporta l'errore se non e' zero
                    '             TOS=PFA di parole da eliminare
                    DUP           Duplica
                    FENCE @ U<    Flag vero se PFA e' sotto FENCE
```

```
                    LIT 0015      Errore numero 21
                    ?ERROR        Errore se TOS e' vero
                    DUP NFA       Duplica PFA e sostituisce NFA
                                  per la copia
                    DUP !         DP=NFA
                    LFA @         Converte PFA a NFA e legge
                                  l'associazione
                    CURRENT @ !   Pone l'associazione alla
                                  locazione definita da CURRENT
```

Notate che non occorrono cancellazioni effettive. Poiche' il puntatore e' stato modificato, le nuove parole verranno scritte sopra quelle vecchie.

```
352  INDEX          CLS           Cancella lo schermo
                    1+ SWAP       Inverte i parametri
                                  incrementando la fine
                    (DO)
353                 CR            Newline
                    13 .R         Scrive il numero dello schermo
                                  in campo a tre spazi
                    SPACE         Somma uno spazio
                    0 I .LINE     Visualizza la linea 0 dello
                                  schermo I
                    ?TERMINAL     Pone a vero se BREAK
                    0BRANCH 0004  Va a 354 se non e' BREAK
                    LEAVE         Termina il ciclo
                    (LOOP) FFE6   Ciclo a 354
354  TRIAD          CLS           Cancella lo schermo
                    3 /           Divide il numero di schermo per
                                  3
                    3 *           Lo moltiplica per 3
                    3 OVER +      Stack: X X+3
                    SWAP          Stack: X+3 X
                    (DO)
356                 CR            Newline
                    I LIST        Lista lo schermo I
                    ?TERMINAL     TOS=1 se BREAK
                    0BRANCH 0004  Se non c'e' BREAK vai a 357
                    LEAVE         Termina il ciclo
357                 (LOOP) FFF0   Ricorsione a 356
*358 EDITOR                       Esegue il codice a 164
                                  Esegue 225
359    (spazio)                   Associa all'ultima istruzione
                                  nel vocabolario EDITOR
360  WHERE          DUP           Stack: IN BLK BLK
                    B/SCE /       Stack: IN BLK BLK+B/SCR
                    DUP SCR !     Definisce SCR da TOS
                                  Lo stack non viene modificato
                    (.") testo    Scrive "SCR #"
                    DECIMAL .     Scrive il numero dello schermo
                    SWAP C/L      Stack: BLK IN C/L
                    /MOD          Stack: BLK resto quoziente
                    C/L *         Stack: BLK resto linea
                    ROT           Stack: reato linea BLK
```

```
                    BLOCK +       Stack: resto linea+indirizzo
                    CR            Newline
                    C/L TYPE      Scrive la linea
                    CR            Newline (Stack: resto)
                    HERE C@ -     Stack: resto-(HERE)
                    SPACES        Emette TOS spazi
                    LIT 005E      Codice per la freccia all'insu'
                    EMIT          Scrive il carattere
                    EDITOR        Seleziona il vocabolario EDITOR
                    QUIT          Ritorna il controllo all'utente
```

Questa utilissima routine puo' venir chiamata in seguito a un errore per trovare la locazione dell'errore. Essa definisce il modo editor e cancella gli stacks.

Questo e' l'inizio del vocabolario EDITOR. Queste istruzioni sono associate all'inizio del vocabolario FORTH tramite il codice in 227. In VLIST, con selezionato l'EDITOR, appare prima il vocabolario EDITOR, cui segue il vocabolario FORTH.

```
361 #LOCATE         R# @          Legge la locazione del cursore
                    C/L /MOD      Stack: colonna linea
362 #LEAD           #LOCATE       Vedi sopra
                    LINE          Stack: colonna indirizzo
                    SWAP          Stack: indirizzo colonna
363 #LAG            #LEAD         Vedi sopra
                    DUP           Stack: ind colonna colonna
                    >R            TOS in TORS
                    +             Stack: ind+colonna
                    C/L R>        Stack: ind+colonna C/L colonna
                    -             Stack: ind+colonna C/L-colonna
364 -MOVE           LINE          Stack: ind1 ind2
                                  (TOS=indirizzo della linea)
                    C/L           Stack: ind1 ind2 C/L
                    CMOVE         Copia una linea da ind1 a ind2
                    UPDATE
365 H               LINE PAD 1+   Stack: ind-linea PAD+1
                    C/L DUP       Stack: ind-linea PAD+1 C/L C/L
                    PAD C!        Stack: prima locaz. PAD
                                  definita a C/L
                    CMOVE         Copia una linea da ind-linea a
                                  PAD
366 E               LINE C/L      Stack: ind C/L
                    BLANKS        Riempie la linea con spazi
                    UPDATE
367 S               DUP 1 -       Stack: Linea Linea-1
                    LIT 000E
                    (DO)
368                 I LINE        Indirizzo della linea I
                    I 1+          I+1
                    -MOVE         Copia la linea I+1
                    LIT -1
                    (+LOOP) FFF0  Vai a 368
                    E             Cancella la linea
369 D               DUP           Stack: Linea Linea
```

```
                    H               Copia la linea in PAD
                    LIT 000F
                    DUP ROT         Stack: 15 15 Linea
                    (DO)
370                 I 1+ LINE       Indirizzo della linea I+1
                    I
                    -MOVE           Copia la linea I+1 a linea I
                    (LOOP) FFF4     Vai a 360
                    E               Cancella la linea 15
371 M               R# +!           Aggiunge TOS alla posizione del
                                    cursore
                    CR SPACE        Newline spazio
                    #LEAD TYPE      Scrive il testo dal cursore
                    LIT 005F        Codice per la sottolineatura
                    EMIT            Scrive il carattere
                    #LAG TYPE       Scrive il testo dal cursore a
                                    fine linea
                    #LOCATE .       Scrive il numero
                    DROP            Scarta la colonna
372 T               DUP             Stack: Linea Linea
                    C/L *           Stack: Linea Linea*C/l
                    R# !            Definisce la posizione del
                                    cursore all'inizio della linea
                    DUP             Stack: Linea Linea
                    H               Copia la linea in PAD
                    O M             Scrive la linea (cursore alla
                                    estremita' sinistra)
373 L               SCR @           Legge il numero dello schermo
                                    corrente
                    LIST            Lista quello schermo
                    O M             Scrive la linea del cursore
374 R               PAD 1+ SWAP     Stack: PAD+Linea
                    -MOVE           Copia la linea da PAD alla
                                    linea specificata
375 P               1 TEXT          Mette il testo in PAD
                    R               Copia il testo alla linea
                                    specificata
376 I               DUP             Stack: Linea Linea
                    S               Fa scivolare di una linea
                    R               Copia la linea da PAD
```

Notate che R e I non sono uniche. Mentre l'EDITOR e' effettivo, vengono sostituiti i significati FORTH. Tuttavia, l'R originale viene usato in diverse definizioni EDITOR.

```
377 TOP             0 R# !          Mette a 0 la posiz. del cursore
378 CLEAR           SCR !           Definisce lo schermo da TOS
                    LIT 0010 0      Sedici iterazioni
                    (DO)
379                 I E             Cancella la linea I
                    (LOOP) FFFA     Va a 379
380 COPY            B/SCR *         Stack: SCR1 SCR1*B/SCR
                    OFFSET @ +      Somma OFFSET
                    SWAP            Stack: SCR2*B/SCR+OFFSET SCR1
```

```
                B/SCR *         Stack: SCR2*B/SCR+OFFSET
                                SCR1*B/SCR (= A B)
                B/SCR OVER +    Stack: A B B+B/SCR
                SWAP            Stack: A B+B/SCR B
                (DO)
381             DUP             Stack: A A
                I BLOCK         Indirizzo del blocco I
                2 - !           Scrive A a ind-2
                1+              Stack: A+1
                UPDATE
                (LOOP) FFEE     Ciclo a 381
                DROP            Scarta A
                FLUSH           Copia il buffer aggiornato al
                                disco RAM
382  -TEXT      SWAP            Stack: ind1 ind2 cont
                -DUP            Duplica TOS se non-zero
                0BRANCH 002A    Se e' zero vai a 386
                OVER +          Stack: Ind1 Ind2 Ind2+Count
                SWAP            Stack: Ind1 Ind2+Count Ind2
                (DO)
283             DUP             Stack: Ind Ind
                C@ I C@ -       Stack: Ind (Ind)-(I)
                0BRANCH 000A    Se zero vai a 384
                0=              Flag vero se zero
                LEAVE
                BRANCH 0004     Vai a 386
384             1+              Incrementa l'indirizzo
385             (LOOP) FFE6     Ciclo a 383
                BRANCH 0006     Fine
386             DROP 0=         Scarta l'indirizzo mette un
                                flag falso
387  MATCH      >R >R           Stack: ind1 cont1
                                (ind2 cont2 in RS)
                2DUP
                R> R>           Stack: ind1 cont1 ind1 cont1
                                ind2 cont2
                2SWAP           Stack: ind1 cont1 ind2 cont2
                                ind1 cont1
                OVER +          Stack: ind1 cont1 ind2 cont2
                                ind1 ind1+cont1
                SWAP            Stack: ind1 cont1 ind2 cont2
                                ind1+cont1 ind1
                (DO)
388             2DUP            Stack: ind1 cont1 ind2 cont2
                                ind2 cont2
                I - TEXT        Stack: ind1 cont1 ind2 cont2
                                flag
                0BRANCH 001A    Se e' falso vai a 389
                >R 2DROP R>     Scarta 20S,30S
                - I SWAP -
                0 SWAP 0 0
                LEAVE           Termina il ciclo
389             (LOOP) FFDC     Ciclo a 388
```

```
                    2 DROP SWAP
                    0=              Inverte il flag
                    SWAP
390  1LINE          #LAG PAD        Stack: cursore a EOL PAD
                    COUNT           Stack: cursore a EOL PAD cont
                    MATCH
                    R# +!           Aggiorna la posizione del
                                    cursore
391  FIND           LIT 03FF
                    R# @ <          Vero se la posizione del
                                    cursore e' minore di 03FF
                    0BRANCH 0012    Se e' zero vai a 392
                    TOP             Azzera la posizione del cursore
                    PAD HERE
                    C/L 1+
                    CMOVE           Copia da PAD a HERE, C/L+1
                                    bytes
                    0 ERROR         Riporta l'errore 0
392                 1LINE           Vedi sopra
                    0BRANCH FFDE    Se e' zero vai a 391
394  DELETE         >R #LAG         Stack: cursore a EOL, n a TORS
                    + R -           Stack: cursore a EOL-n
                    #LAG            Stack: cursore a EOL-n
                                    cursore a EOL
                    R MINUS         Somma allo stack -n
                    R# +!           Somma -n alla posizione del
                                    cursore
                    #LEAD +
                    SWAP
                    CMOVE
                    R> BLANKS
                    UPDATE
395  N              FIND            Ricerca
                    0 M             Visualizza
396  F              1 TEXT          Testo in PAD
                    N               Trova il testo
398  B              PAD C@          Legge la lunghezza da PAD
                    MINUS           Nega
                    M               Somma alla posizione del
                                    cursore
399  X              1 TEXT          Testo in PAD
                    FIND            Localizza
                    PAD C@          Legge la lunghezza da PAD
                    DELETE          Cancella gli ultimi n caratteri
                    0 M             Visualizza la linea
400  TILL           #LEAD +         Linea+colonna
                    1 TEXT          Testo in PAD
                    1LINE
                    0=              Inverte il flag
                    0 ?ERROR        Errore se e' vero
                    #LEAD +         Linea+colonna
                    SWAP -
                    DELETE
```

```
                0 M             Visualizza la linea
401  C          1 TEXT          Testo in PAD
                PAD COUNT       Modifica il riferimento
                #LAG
                ROT OVER        Stack: da  a b c  a  b c a c
                MIN >R
                R R# +!         Somma TORS alla  posizione  del
                                cursore
                R - >R
                DUP HERE R
                CMOVE
                HERE #LEAD
                + R>
                CMOVE
                R>
                CMOVE
                UPDATE
                0 M             Visualizza la linea
```

Questo completa il vocabolario EDITOR. Quando il vocabolario e' inabilitato, il dizionario ricerca l'inizio con la parola C. La parola NEXT piu' sotto, e' associata a WHERE (360)

```
402                 NEXT        Indirizzo di  ritorno  costante
                                associato a 3
403                 PUSHHL      Indirizzo di  ritorno  costante
                                associato a 2
404                 PUSHDE      Indirizzo di  ritorno  costante
                                associato a 1
405  INP                        POP HL.PUSH BC.BC=HL.IN A,(C).
                                POP BC.H=0.L=A.PUSH HL.
                                Vai a NEXT1
406  OUTP                       POP HL.POP DE.PUSH BC.BC=HL.
                                A=E.OUT (C),A.POP BC.
                                Vai a NEXT1
407  SCREEN                     POP HL.POP DE.PUSH BC.PUSH IX.
                                C=E.B=L.CALL 2538.CALL 2BF1.
                                A=(DE).H=0.L=A.POP IX.POP BC.
                                PUSH HL.Vai a NEXT1
408  AT             ABS DUP     Stack: linea colonna linea
                    LIT 001F >
                    0BRANCH 0008 Se colonna non maggiore  di  31
                                vai a 409
                    2DROP       Cancella lo Stack
                    BRANCH 0022 Fine
409                 SWAP ABS DUP Stack: colonna linea linea
                    LIT 0015 >
                    0BRANCH 0008 Se linea non maggiore di 21 vai
                                a 410
                    2DROP       Cancella lo stack
                    BRANCH 000C Fine
410                 LIT 0016
                    EMIT        Emetti 22
                    EMIT        Visualizza la linea
```

```
                EMIT            Visualizza la colonna
411  BORDER                     POP HL.PUSH BC.A=L.CALL 2297.
                                POP BC.Vai a NEXT1
412  BLEEP                      POP HL.POP DE.PUSH BC.PUSH IX.
                                CALL 03B5.POP IX.POP BC.
                                Vai a NEXT1.
413  PAPER      ABS DUP
                LIT 0009 >
                0BRANCH 0008    Se il parametro non e' maggiore
                                di 9 vai a 414
                DROP            Cancella lo stack
                BRANCH 0088     Fine
414             DUP
                LIT 0009 =
                0BRANCH 001A    Se il parametro non e' 9 vai  a
                                415
                LIT 5C91        Indirizzo di PFLAG
                C@              Legge PFLAG
                LIT 0080 OR     Mette a 1 il bit 3
                LIT 5C91 C!     Scrive in PFLAG
                DROP            Scarta il parametro
                BRANCH 0064     Fine
415             DUP
                LIT 0008 =
                0BRANCH 001A    Se il parametro non e' 8 vai  a
                                416
                LIT 5C8E        Indirizzo di MASKP
                C@              Legge MASKP
                LIT 0038 OR     Mette a 1 i bits 3, 4 e 5
                LIT 5C8E        Indirizzo di MASKP
                DROP            Scarta il parametro
                BRANCH 0040     Fine
416             LIT 0008 *      Moltiplica il parametro per 8
                LIT 5C8D C@     Legge ATTRP
                LIT 00C7 AND    Azzera i bits 3, 4 e 5
                OR              OR dei parametri
                LIT 5C8D C!     Scrive in ATTRP
                LIT 5C91 C@     Legge PFLAG
                LIT 007F AND    Azzera il bit 7
                LIT 5C91 !      Scrive in PFLAG
                LIT 5C8E C@     Legge MASKP
                LIT 00C7 AND    Azzera i bits 3, 4 e 5
                LIT 5C8E !      Scrive in MASKP
417  ATTR                       POP HL.POP DE.PUSH BC.PUSH IX.
                                C=E.B=L.CALL 2583.CALL 1E94.
                                H=0.L=A.POP IX.POP BC.PUSH HL.
                                Vai a NEXT1
418  POINT                      POP HL.POP DE.PUSH BC.PUSH IX.
                                C=E.B=L.A=L.Se A non e'  minore
                                di HOH allora A=AFH,B=A.
                                CALL 22CE.CALL 1E94.H=0.L=A.
                                POP IX.POP BC.PUSH HL.
                                Vai a NEXT1.
```

```
419  INK          ABS DUP
                  LIT 0009 >
                  0BRANCH 0008 Se il parametro non e' maggiore
                               di 8 vai a 420
                  DROP         Scarta il parametro
                  BRANCH 0082  Fine
420               DUP
                  LIT 0009 =
                  0BRANCH 001A Se il parametro non e' 9 vai  a
                               421
                  LIT 5C91 C@  Legge PFLAG
                  LIT 0020 OR  Mette a 1 il bit 5
                  LIT 5C91 C!  Scrive in PFLAG
                  DROP         Scarta il parametro
                  BRANCH 005E  Fine
421               DUP
                  LIT 0008=
                  0BRANCH 001A Se il parametro non e' 8 vai  a
                               422
                  LIT 5C8E C@  Legge MASKP
                  LIT 0007 OR  Mette a 1 i bits 0,1 e 2.
                  LIT 5C8E C!  Scrive in MASKP
                  DROP         Scarta il parametro
                  BRANCH 003A  Fine
422               LIT 5C8D C@  Legge ATTRP
                  LIT 00F8 AND Azzera i bits 0, 1 e 2.
                  OR           OR del parametro
                  LIT 5C8D C!  Scrive in ATTRP
                  LIT 5C91 C@  Legge PFLAG
                  LIT 00DF AND Azzera il bit 5
                  LIT 5C91 !   Scrive in PFLAG (una parola)
                  LIT 5C8E C@  Legge MASKP
                  LIT 00F8 AND Azzera i bits 0, 1 e 2.
                  LIT 5C8E !   Scrive in MASKP (una parola)
423  FLASH        0BRANCH 0018 Se il parametro e' 0 vai a 424
                  LIT 5C8D C@  Legge ATTRP
                  LIT 0080 OR  Mette a 0 il bit 7
                  LIT 5C8D !   Scrive in ATTRP
                  BRANCH 0014  Fine
424               LIT 5C8D C@  Legge ATTRP
                  LIT 007F AND Azzera il bit 7
                  LIT 5C8D !   Scrive in ATTRP (una parola)
425  BRIGHT       0BRANCH 0018 Se il parametro e' 0 vai a 426
                  LIT 5C8D C@  Legge ATTRP
                  LIT 0040 OR  Mette a 1 il bit 6
                  LIT 5C8D !   Scrive in ATTRP (una parola)
                  BRANCH 0014  End
426               LIT 5C8D C@  Legge ATTRP
                  LIT 00BF AND Azzera il bit 6
                  LIT 5C8D !   Scrive in ATTRP
427  GOVER        0BRANCH 0016 Se il parametro e' 0 vai a 428
                  LIT 5C91 C@  Legge PFLAG
                  2 OR         Mette a 1 il bit 1
```

```
                    LIT 5C91 !    Scrive in PFLAG
                    BRANCH 9914   Fine
428                 LIT 5C91 C@   Legge PFLAG
                    LIT 00FD AND  Azzera il bit 1
                    LIT 5C91 !    Scrive in PFLAG
429 INVERSE         0BRANCH 0018  Se il parametro e' 0 vai a 430
                    LIT 5C91 C@   Legge PFLAG
                    LIT 0008 OR   Mette a 1 il bit 3
                    LIT 5C91 !    Scrive in PFLAG
                    BRANCH 0014   Fine
430                 LIT 5C91 C@   Legge PfLAG
                    LIT 00F7 AND  Azzera il bit 3
                    LIT 5C91 !    Scrive in PFLAG
431 NOT             0=            Sinonimi
432 I'                            HL=(RSP)+2.DE=(HL).PUSH DE.
                                  Vai a NEXT1
433 J                             HL=(RSP)+4.DE=(HL).PUSH DE.
                                  Vai a NEXT1
434 2CONSTANT       CREATE
                    SMUDGE
                    HERE 2!       Scrive un numero doppio
                    LIT 0004
                    ALLOT         Riserva ulteriori quattro bytes
                    (;CODE)       Associazione al codice seguente
435                               INC DE.Scambia DE,HL.HL=HL+2.
                                  DE=(HL).PUSH DE.HL=HL-2.DE=(HL)
                                  .PUSH DE.Vai a NEXT1.
436 2VARIABLE       2CONSTANT
                    (;CODE)       Associazione al codice seguente
437                               INC DE.PUSH DE.Vai a NEXT1.
438 U.R             >R 0 R>       Rende 20S un numero doppio
                                  (senza segno)
                    D.R
```

Vocabolario FORTH

Il numero di ogni parola e' un'associazione all'Appendice A

!	77	2+	125
!CSP	146	2!	79
#	322	2@	76
#>	320	2CONSTANT	434
#BUFF	280	2DRCP	347
#S	324	2DUP	717
'	302	2OVER	439
(	229	2SWAP	348
(.")	173	2VARIABLE	438
(;CODE)	160	:	80
(+LOOP)	13	;	82
(ABORT)	205	;CODE	161
(DO)	14	;S	56
(FIND)	19	<	132
(LINE)	258	<#	319
(LOOP)	9	<BUILDS	162
(NUMBER)	198	=	131
(TAPE)	340	>	135
*	251	>R	58
*/	256	?	329
*/MOD	255	?COMP	149
+	63	?CSP	152
+!	72	?ERROR	147
+-	241	?EXEC	150
+BUFF	281	?LOADING	153
+LOOP	309	?PAIRS	151
+ORIGIN	100	?STACK	215
,	128	?TERMINAL	34
-	130	@	74
-->	301	ABORT	233
-DUP	138	ABS	243
-FIND	204	AGAIN	312
-TRAILING	169	ALLOT	127
.	328	AND	49
."	174	AT	408
.CPU	339	ATTR	417
.LINE	259	B/BUFF	98
.R	326	B/SCR	99
/	253	BACK	303
/MOD	252	BASE	118
0	90	BEGIN	304
1	91	BL	94
2	92	BLANKS	129
3	93	BLEEP	412
0<	62	BLK	110
0=	61	BLOCK	290
0BRANCH	8	BRANCH	7
1+	124	BRIGHT	425

cabolario EDITOR

"FORTH PER SPECTRUM" è un aiuto essenziale per chiunque desideri scoprire il vero potenziale del FORTH sul proprio SPECTRUM ed è l'ideale sia per il principiante che per il programmatore avanzato in quanto propone esempi e spiegazioni molto esaurienti.
La popolarità del FORTH come linguaggio alternativo, ché combina la facilità di un linguaggio ad alto livello e la velocità del codice macchina, ha indotto a renderlo disponibile per l'utente Spectrum.
Questo libro è stato progettato per fornire in dettaglio le tecniche che consentiranno al programmatore di comprendere interamente la validità del FORTH.
Oltre ad affrontare in modo approfondito il fig-FORTH ABERSOFT, questo libro tratta delle tecniche specifiche di programmazione e della stesura di programmi mediante molti esempi.
Inoltre vengono affrontati vari argomenti come: l'aritmetica su FORTH, la manipolazione di stringhe e matrici, la definizione di nuove parole del dizionario, alcune particolarità del FORTH per lo Spectrum e le caratteristiche essenziali del FORTH ABERSOFT.

Cod. 9005

L. 15.000

FORTH PER SPECTRUM

di DON THOMASSON

edizioni JCE