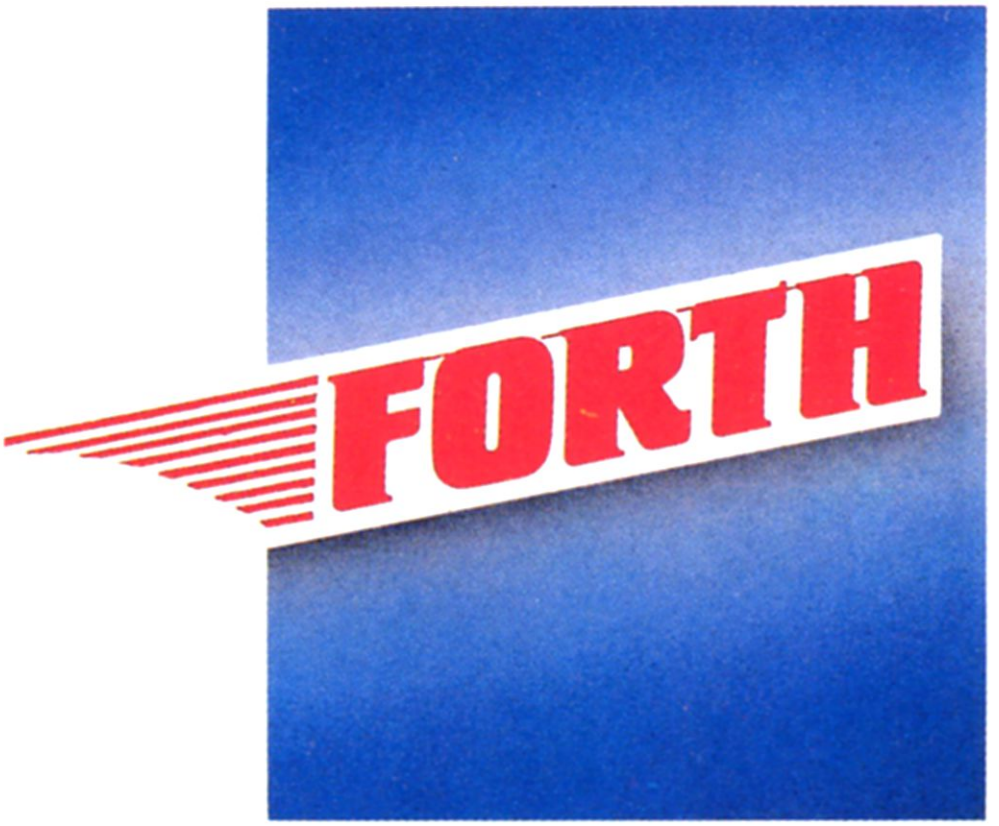

AMSTRAD

FORTH

SOFT 188 (Cinta)
SOFT 1188 (Disco)
Para el Amstrad CP464/664

Publicado por AMSFOFT, una división de
Amstrad Consumer Electronics plc
Bretwood House
169 Kings Road
Brentwood
Essex

Primera edición 1985

Contenido

Capítulo 1
Introducción

Comenzando

Este manual no pretende ser un tutor de FORTH; hay mucha bibliografía disponible en el mercado que llena este cometido más que adecuadamente, sin embargo, el Capítulo 2 brinda una breve pero valiosa guía de los fundamentos de FORTH, además de las extensiones presentadas para el Amstrad CPC464. Para el novato, un buen libro sería 'Starting FORTH' ('Comenzando FORTH') por Leo Brodie, aunque describe el FORTH-79 y no fig-FORTH. Para el usuario más aventajado una referencia útil es 'The Systems Guide to fig-FORTH' ('La Guía de Sistemas del fig-FORTH') por C. H. Ting.

FORTH es un lenguaje que combina las características de los lenguajes de alto nivel con la velocidad del lenguaje-máquina. El intérprete de FORTH interpreta cada línea de comandos buscando en un diccionario de palabras interno que él entiende y actúa en consecuencia. Algunas palabras permiten la creación de otras nuevas, las que, a su vez, son la base de palabras de un nivel aún más alto. Eventualmente se forma un programa y puede ser llamado con una sola palabra. Este desarrollo gradual permite la revisión de secciones de un programa mucho mejor que BASIC.

Cargando

1) Cinta

Para cargar el sistema básico en su Amstrad CPC464, teclee

```
RUN "FORTH"
```

Pulse PLAY en el cassette y, a continuación, pulse la tecla ENTER. Cuando el compilador halla cargado verá el mensaje:

```
Amstrad CPC464 fig-FORTH 1.1B
(c)Abersoft:1984
```

y el cursor normal.

2) Disco

Para cargar el sistema básico en su sistema CP/M del Amstrad CPC464, teclee:

```
A>FORTH <nombre del fichero>
```

donde 'nombre del fichero' es el nombre del que será usado para guardar pantallas fuente del editor. Si no se diera un nombre, el programa pondrá FORTH.FIG por defecto. Cualquiera que sea el nombre que se dé, se le añadirá la extensión .FIG. (Este fichero es el primero que se utiliza durante una sesión con el compilador; si durante la programación se requiere otro fichero .FIG se usará FILE <nombre del fichero>, por ejemplo: FILE EDITOR). Esto cierra el fichero en uso y abre uno nuevo para I/O. Observe que no se permite extensión.

Cuando el compilador haya cargado, verá el mensaje:

```
Amstrad CPC464 CP/M fig-Forth 1.1B
(c)Abersoft:1984
```

y el cursor normal.

Para probar que el sistema funciona, pulse ENTER y el sistema deberá responder

```
Ok
```

Ya está listo para comenzar a programar en FORTH.

Capítulo 2
Guía de FORTH para Principiantes

El stack (La pila)

Una de las mayores diferencias entre FORTH y la mayor parte de los otros lenguajes de alto nivel, es que el primero utiliza Notación Polaca Inversa (RPN). Normalmente el operador + se coloca entre los números que quiere sumar (p.e.: 4 + 7). En RPN, por el contrario, el operador viene después de los números (p.e.: 4 7 +). (Observe que necesita insertar un espacio entre el 4, el 7 y el signo +). Esto se debe a que se utiliza una 'pila' para guardar los números, al evaluar las expresiones. (En caso de que no sepa qué es una 'pila', imagine un montón de platos. El último plato apilado, al estar encima del todo, es el primero que se quitará después). Cuando FORTH encuentra un número se coloca en la pila. Cuando encuentra un operador, toma la cantidad necesaria de números de la pila, y vuelve a colocar el resultado en ésta. La palabra punto . toma un número de arriba de la pila y lo imprime.

La línea en BASIC

```
PRINT (3+7)*(8+2)
```

es, en FORTH

```
3 7 + 8 2 + * .
```

(observe que los espacios son importantes).

Después de ejecutada, la pila se convierte en:

3	(3) la 'cima' de la pila
7	(3,7)
+	(10)
8	(10,8)
2	(10,8,2)
+	(10,10)
*	(100)
.	() se imprime 100

La única forma de familiarizarse con RPN es practicar. Probablemente, lo mejor que puede hacer ahora, si es novato programando, es sentarse al teclado y probar con algunos ejemplos. Después de un rato, RPN será tan fácil como la aritmética normal y no tendrá problemas en la programación. Puede que a veces reciba el mensaje ? MSG # 1 cuando trate de hacer algo; ésto simplemente significa que ha tratado de usar más información de la que tiene en la pila.

Otras palabras que se explican en la guía de referencia son:

+ - / */ */MOD /MOD MOD MAX MIN AND OR XOR MINUS +- 1+ 2+ .

Hay palabras que alteran el orden de los números en la pila.

DUP duplica el número de arriba

DROP desecha el número de arriba

SWAP intercambia los dos primeros números de arriba

ROT rota los tres primeros números de arriba, colocando el tercero en la cima

OVER copia el segundo valor en la cima como nuevo primer valor

Por ejemplo:

1 DUP . . imprime 1 1 ok

1 2 DROP . imprime 1 ok

1 2 SWAP . . imprime 1 2 ok

Vea también: S0 SP@

Variables y Constantes

Una variable en FORTH debe ser creada explícitamente antes de ser usada, utilizando la palabra VARIABLE.

```
3 VARIABLE A1
```

creará la variable A1 con el valor inicial 3. Cualquier secuencia de caracteres alfanuméricos será considerada un nombre.

```
6 A1 !
```

guarda el 6 en A1 una vez que A1 ha sido definida como una variable. (Esto equivale a A1=6 en BASIC).

```
A1 @ (puede escribirse también A1 ?)
```

buscará el contenido de A1 y lo imprimirá. La palabra A1 de hecho coloca la dirección de A1 en la pila. ! toma una dirección de la pila y después toma un número de la pila y la guarda en esa dirección. La palabra @ toma una dirección de la pila y la reemplaza con el contenido de la memoria en esa dirección. La palabra ? es el equivalente de @ . -para aquellos que son perezosos-. Vea también la palabra + !.

Una constante en FORTH es un número fijo al que se le da un nombre, usando la palabra CONSTANT.

```
10 CONSTANT TEN
```

Crea una constante, TEN, con valor 10. De ahora en adelante, cada vez que aparezca la palabra TEN, se colocará al número 10 en la pila, por tanto,

```
TEN .
```

imprimirá:

```
10 Ok
```

Observe que ! y @ no son necesarias con constantes, y de hecho ! no puede ser usado para cambiar una constante, porque CONSTANT es una constante.

Entre las variables predefinidas por el sistema está BASE. Esta contiene el número base actual utilizado para entrada y salida (input/output). HEX guarda 16 (base 16) en BASE, fijando el modo hexadecimal. DECIMAL guarda 10 (base 10) en BASE, fijando el modo decimal.

Tipos de números

FORTH es un lenguaje sin tipos de números. Por ejemplo, no existe distinción entre un número y un carácter. Lo que hay en la pila lo toma una palabra y ésta lo interpreta como considere conveniente. Algunas de las interpretaciones más comunes, que no son la normal de número con signo de 16 bits (entre -32768 y 32767), son:

Sin signo:

entre 0 y 65535. La palabra U . actúa como . pero supone que el número no tiene signo.

Doble precisión:

Los dos números, arriba del todo en la pila, se interpretan como un número de 32 bits, bien con signo o sin él. El número de la cima se toma como los 16 bits más altos. El segundo se toma como los 16 bits más bajos. Para poner un número de 32 bits en la pila, escriba el número con un punto decimal en cualquier parte de él. La variable del sistema DPL contiene el número de dígitos que siguen al punto decimal, por si fuera importante.

```
70.000 . . DPL @ .
```

Imprimirá 1 4464 3 ok. El número de 32 bits 70000 (no 70) se coloca en la pila como dos números. El primer . imprime la mitad superior como un número con signo. La mitad inferior se imprime después (observe que 70000 = 1 x 2 ↑ 16 + 4464). El contenido de DPL es 3, ya qe hay 3 dígitos después del punto decimal.

Byte:

A veces sólo se requieren números de 8 bits, por ejemplo, para representar un carácter. Se representan en la pila por números de 16 bits con los 8 más altos a cero, C @ y C ! buscan y guardan un sólo byte a la vez.

Flags (banderines)

Para tomar decisiones son necesarios los valores verdadero y falso. Por ejemplo:

```
2 3 = .
```

imprime **Ø (falso),** mientras que

```
3 3 = .
```

imprime 1 (verdadero). De hecho, cualquier número distinto de cero es considerado como verdadero. (Por tanto, - puede ser usado como < > , ya que x - y = 0 (falso) sólo cuando x = y.) Otras comparaciones son:

< U< 0< (Equivalente a 0 <) y 0= (equivalente a 0 =).

NOT cambia un barderín de verdadero a falso y . σ viceversa (es en realidad equivalente a Ø =.).

Modos de FORTH

FORTH tiene dos modos de trabajo. Uno es el modo intérprete y el otro es el modo compilado.

En el modo intérprete FORTH toma lo que se dé al ordenador y trata de ejecutarlo inmediatamente. Si escribe

```
4 7 + .
```

el ordenador dará respuesta inmediata con

```
11 Ok
```

Sin embargo, en el modo compilado, FORTH lo almacena en su diccionario y lo usa después. Por ejemplo, si sólo quería introducir la suma anterior, y ver el resultado más adelante (algo extraño de hacer) podría definir una nueva palabra, a saber:

```
: STRANGE 4 7 + . ;
```

(: es la palabra usada por FORTH que significa comienzo de compilación ; es la palabra que significa fin de compilación). Si después escribe:

```
STRANGE
```

y ENTER, el ordenador le dará como respuesta:

```
11 Ok
```

Una vez que una palabra sea definida de esta forma, puede ser usada en otras definiciones:

```
: TOOSTRANGE STRANGE STRANGE ; TOOSTRANGE
```

compila la palabra TOOSTRANGE, que ejecutará STRANGE dos veces, imprimiendo:

```
11 11 Ok
```

Observe que, en lo que respecta a FORTH, una vez que una palabra es definida usando : . . . ; ésta se convierte en parte del lenguaje y puede ser usada exactamente de la misma forma que cualquier otra palabra, como SWAP y DROP. Al definir nuevas palabras, trate de darles un nombre que signifique lo que hacen. En medio de un programa largo, una palabra definida como CENT-FAHR es bastante obvia, mientras que CF no lo sería. (Como ejemplo de una definición FORTH más útil que STRANGE, aquí tenemos CENT-FAHR

```
: CENT-HAHR   (espera un sólo número en la pila)
9 *           (multiplica por 9)
5 /           (divide entre 5)
32 +          (suma 32)
.             (imprime la respuesta)
;             (definición terminada)
```

ahora, al escribir 10 CENT-FAHR ENTER aparecerá en pantalla

```
10 CENT-FAHR 50 Ok
```

Estructuras de Control

La estructura FORTH IF . . . ENDIF (o IF. . .ELSE ...ENDIF) permite la ejecución condicional (sólo) dentro de una definición. El equivalente de IF A >2 THEN PRINT "TOO BIG" en BASIC, es

```
: TEST A @ 2 > IF . " TOO BIG" ENDIF ;
```

A @ 2 > pone un banderín en la pila significando si A es mayor que 2. IF quita el banderín. Si el banderín es verdadero (distinto de cero) el código que sigue se ejecuta. La palabra . " imprime todo lo que halle hasta las siguientes comillas ". (Observe que debe haber un espacio después de . " para que sea reconocido debidamente). Si el banderín es falso (cero) el código hasta ENDIF se pasa por alto.

En el caso de IF...ELSE...ENDIF, si el banderín es verdadero el código entr IF y ELSE se ejecuta y el código entre ELSE y ENDIF se pasa por alto. Si e banderín es falso, sólo el código ELSE...ENDIF (y lo que sigue a ENDIF, com de costumbre) se ejecuta. Dentro de cualquier IF...ENDIF puede haber otr similar.

El

```
FOR A=1 TO 10: PRINT A:NEXT A     de BASIC
```

es el

```
: TEST 11 1 DO I . LOOP ;         de FORTH
```

Cuando se ejecuta TEST, DO toma dos números de la pila. El número de arriba (1 aquí) es el valor inicial. El segundo número de la pila (11) es e límite. I copia el índice del bucle (A en el programa de BASIC) en la pila donde . lo imprime. LOOP incrementa el índice en 1. Si se llega al límite o se sobrepasa, la ejecución continúa normalmente, de lo contrario regresa a DO. Por lo que el límite es 11 a fin de que cuente hasta 10. Observe que el bucle se ejecuta una vez, no importa cuál sea el límite. n + LOOP, en vez de LOOP, es el equivalente en FORTH del STEP n en BASIC. LEAVE cambia el límite de manera que se saldrá del bucle tan pronto se llegue a LOOP. J dá el valor del contador del bucle exterior si hay una estructura de bucles anidados. Por ejemplo:

```
: DEMO CR
        4 1 DO
          9 7 DO
            I J . . CR
          LOOP
        LOOP
;
```

imprimirá

```
1 7
1 8
2 7
2 8
3 7
3 8
```

Otra clase de bucle es BEGIN...banderín UNTIL

```
: TEST BEGIN  ...  A @ 2 > UNTIL ;
```

es el equivalente de:

```
10 REMark BEGIN
20 ...
30 IF NOT (A>2) THEN GOTO 10
```

por ejemplo, se ejecutará una y otra vez hasta que A>2.

Esta es una instrucción muy útil cuando necesitamos repetir un bucle una y otra vez hasta que se pulse la tecla ESC . Puede hacerse de esta forma:

```
: TEST BEGIN . ' LOOPING' CR ?TERMINAL UNTIL ;
```

El ?TERMINAL deja un banderín de verdadero en la pila si se pulsa ESC y falso si no se pulsa.

Si se desea que un bucle continúe indefinidamente, puede usarse BEGIN ... AGAIN. No tiene condiciones y sería de mucha utilidad al escribir un programa de juegos, por ejemplo:

```
: GAME
        TITLES
        BEGIN
                PLAY-GAME
                END-MESSAGE
        AGAIN
;
```

Esto también demuestra cómo deben ser organizados los programas, utilizando la palabra final para llamar a un programa, consistente en sólo unas cuantas palabras, construídas y probadas.

El problema con las estructuras condicionales explicadas es que son ejecutadas por lo menos una vez, ya que la prueba se lleva a cabo al final del bucle. Ciertos programas requerirán que la prueba se lleve a cabo al comienzo del programa, lo que se hace con la estructura BEGIN...banderín WHILE...REPEAT. Por ejemplo:

```
: DEMO
        BEGIN DUP 4 > WHILE
        DUP . 1 - REPEAT
        DROP
;
```

Si se escribe 7 DEMO, dará como resultado 7 6 5; si se escribe 3 DEMO no habrá resultado.

Uno de los comandos más útiles en un lenguaje como Pascal es CASE. Permite probar un número para valores diferentes y ejecutar diferentes procedimientos con cada valor. En FORTH estándar este comando está disponible sólo usando IF ... IF ... IF ENDIF ENDIF ENDIF y puede ser muy tedioso. Una nueva estructura en FORTH Abersoft se llama estructura CASE. Se usa así:

.....(instrucciones que dejan un sólo valor en la pila)

```
CASE
     8 OF . ' Este es 8' CR ENDOF
     12 OF . ' Este es 12' CR ENDOF
     99 OF . ' Este es 99' CR ENDOF
ENDCASE
```

Esta estructura brinda una forma más legible y menos propensa a cometer errores al hacer decisiones múltiples. En el ejemplo anterior, si en la pila se dejó el 99, se imprimirá 'Este es 99'. Cualquier otro valor distinto de 8, 12 ó 99 no produciría resultado alguno.

I/O por Teclado y Pantalla

Ya conoce la palabra . " que imprime un mensaje en la pantalla. EMIT espera el código ASCII de un carácter en la pila. Después imprime ese carácter.

```
65 EMIT
```

imprimirá la letra A (código ASCII 65)

TYPE se utiliza para imprimir un grupo de caracteres. La variable TIB contiene la dirección del Terminal Input Buffer (Terminal de Entrada del Buffer) donde se almacena lo que teclea. Para teclear los primeros 10 caracteres en el buffer, debe suministrar a TYPE la dirección de comienzo, y el número de caracteres a imprimir.

```
TIB @ 10 TYPE
```

imprimirá

```
TIB @ 10 T
```

SPACES toma una cantidad n de la pila e imprime n espacios.

KEY espera a que se pulse una tecla y coloca el código ASCII de esa tecla en la pila.

EXPECT requiere una dirección y una cantidad, como TYPE. Sin embargo, EXPECT toma el número especificado de caracteres del teclado (o todos los caracteres hasta ENTER) y coloca uno o dos nulos (ASCII 0) al final. La palabra QUERY se define como 'TIB @ 80 EXPECT.

GET es similar a KEY, pero mientras que KEY sólo permite los códigos ASCII 13 y del 32 al 127 y muestra el cursor, GET dá los valores de todas las teclas sin que aparezca el cursor.

Las Extensiones de Amstrad

Estas extensiones se presentan en orden alfabético de la misma forma que la Guía de BASIC del Usuario.

n1 n2 BORDER

Da al borde de la pantalla los colores n1 y n2. Si se quiere que el borde tenga un color invariable, n1 y n2 deben ser iguales.

CLG

Borrar la ventana de gráficos.

n CHAN

Fija la salida de texto por el canal señalado.

n1 n2 DRAW

Dibuja una línea desde la posición actual hasta n1,n2.

n1 n2 DRAWR

Dibujar una línea n1,n2 relativa a la posición actual.

(nnc nnb nna. . .n1c n1b n1a) p n ENT

Fija la envolvente de volumen n según los parámetros n1a etc., siendo p la cantidad de grupos de parámetros, e igual que como se explica en la Guía del Usuario.

(nnc nnb nna . . .n1c n1b n1a) p n ENV

Fija la envolvente de volumen n según los parámetros n1a etc., siendo p la cantidad de grupos de parámetros, e igual que como se explica en la Guía del Usuario.

n GPAPER

Fija el color de papel de gráficos con la tinta n.

n GPEN

Fija el color de la pluma de gráficos con la tinta n.

n1 n2 n3 n4 GWINDOW

Fija la ventana de gráficos en n1 (izqda.), n2 (dcha.), n3 (arriba), n4 (abajo).

n1 n2 n3 INK

Fija la tinta n3 con los colores n1 y n2.

n1 INKEY n2

Comprueba la tecla n1: da el valor -1 si no se pulsa; 0 si sólo se pulsa esa tecla; +32 si se pulsa SHIFT además; + 128 si se pulsa CTRL.

INKEY$ n

Da el valor de la tecla que se pulsa.

n1 JOY n2

Da el valor n2 del joystick n1

n1 n2 LOCATE

Fija el cursor de texto en n1, n2 en el canal de texto vigente.

n MODE

Fija el modo de pantalla en n.

n1 n2 MOVE

Mueve el cursor de gráficos hasta n1,n2.

n1 n2 MOVER

Mueve el cursor de gráficos a n1,n2 relativo a la posición actual.

n1 n2 ORIGIN

Fija la posición de comienzo del cursor de gráficos en n1, n2.

n PAPER

Fija el papel en el canal vigente con la tinta n.

n PEN

Fija la pluma en el canal vigente con la tinta n.

n1 n2 PLOT

Marca el punto n1, n2.

n1 n2 PLOTR

Marca el punto n1, n2 relativo a la posición actual del cursor.

POS n

Da la posición horizontal del cursor de texto en el canal vigente.

n RELEASE

Da salida al sonido por el canal n.

SCREEN n

Da el carácter en la posición actual. Si no hay carácter reconocible da 0.

n1 n2 n3 n4 n5 n6 n7 SOUND

Consulte el Capítulo 6 de la Guía del Usuario. (6.10) Donde n1=M, n2=L, n3=K, n4=J, n5=I, n6=H y n7=G.

n1 n2 SPEEDINK

Fija la velocidad en centelleo de las tintas n1, n2.

n1 n2 SPEEDKEY

Fija el retardo de comienzo y el período de repetición en n1 y n2, respectivamente.

n SPEEDWRITE

Si n es cero fija la velocidad más lenta de la cinta, de lo contrario, la más rápida.

n1 SQ n2

Devuelve en n2, el estado de la cadena de sonidos n1.

n1 n2 n3 n4 n5 n6 n7 n8 n9 SYMBOL

Redefine el carácter n9 según los valores n1-n8 mientras ese carácter permita ser redefinido en SYMBOLAFTER. n8 es la fila superior del carácter y n1 la fila inferior.

n SYMBOLAFTER

Permite a todos los caracteres después de n ser definidos por el usuario por símbolo.

TAG

Fija el cursor de texto, en el canal vigente, al cursor de gráficos.

TAGOFF

Invierte la acción de TAG.

n1 n2 TEST n3

Da el valor de la tinta del punto gráfico n1, n2.

n1 n2 TESTR n3

Da el valor de la tinta del punto gráfico n1, n2 relativo a la posición actual del cursor de gráficos.

VPOS n

Da la posición vertical del cursor de texto en el canal vigente.

n1 n2 n3 n4 WINDOW

Fija la ventana de texto n1(izqda.), n2(dcha.), n3(arriba), n4(abajo) en el canal de texto vigente.

n1 n2 WINDOWSWAP

Intercambia las ventanas de texto n1 y n2.

XPOS n1

Da la posición x del cursor gráfico.

YPOS n1

Da la posición y del cursor gráfico.

Vocabulario

VLIST imprime todas las palabras conocidas del vocabulario vigente. (Un vocabulario es una sub-sección de todo el diccionario. El vocabulario normal se llama FORTH y es elegido usando la palabra FORTH).

FORGET <palabra>

Olvidará todas las palabras definidas desde la palabra en adelante. Algo práctico de hacer es compilar la palabra TASK como primera palabra nueva.

: TASK ;

Si después de compilar más palabras, decide deshacerse de todas ellas

FORGET TASK

lo hará. La variable del sistema FENCE contiene una dirección, debajo de la cual no puede olvidar (FORGET) una palabra. Para proteger las palabras que acaba de compilar escriba HERE FENCE !.

Para crear un nuevo vocabulario MYWORDS

escriba

```
VOCABULARY MYWORDS INMEDIATE
```

La palabras MYWORDS hará que se busque este nuevo vocabulario cuando se interpreten las palabras (la variable del sistema CONTEXT se fija en MYWORDS). Sin embargo, nuevas definiciones se añadirán al vocabulario viejo (la variable del sistema CURRENT todavía apunta a FORTH). Para elegir el nuevo vocabulario como al que se le añadirán nuevas definiciones, escriba

```
MYWORDS DEFINITIONS
```

Esto fija el vocabulario actual al de contexto (hecho MYWORDS por MYWORDS). Para volver a añadir definiciones al vocabulario FORTH, escriba,

```
FORTH DEFINITIONS
```

Algunas palabras, tales como FORTH, son inmediatas. Esto quiere decir que serán ejecutadas aún en modo compilado.

El Disco RAM

Uno de los problemas de usar FORTH en un sistema que no sea de disco, es que, una vez que una palabra ha sido definida, se pierde la fuente original de esa definición. Para una definición larga, ésto es una molestia evidente, si se descubre que no realiza lo que se esperaba de ella. En Abersoft FORTH (versión de cinta) ésto ha sido obviado usando como almacenamiento un disco ficticio de 11K en total. Puede parecer pequeño, pero dado lo compacto de FORTH, se ha podido escribir una aplicación sorprendentemente grande. Por ejemplo, el editor descrito más adelante ocupó solamente 8 páginas de este disco (una página tiene 1024 bytes de longitud, distribuidos en 16 líneas de 64 caracteres) y ésto incluye abundantes comentarios.

Las páginas en el disco están numeradas del 0 al 10; la 0 está reservada para comentarios, y el texto se introduce en el disco por medio del editor que se describirá más adelante. Para compilar un programa del disco-RAM, use

n LOAD (donde n es el número de la página de la primera definición).

si la definición o definiciones abarcan más de una página, las palabras finales de la página deben ser

-- (pronunciado siguiente pantalla)

Para preparar el disco RAM para grabar, el comando

INIT-DISC

deberá ser usado. Esto limpia el área con blancos. Para listar el contenido actual de las páginas del disco, use

n LIST (donde n es la página a listar)

Cuando se ha llenado un disco puede ser grabado en cinta con el comando:

SAVET

Para volver a cargar un disco creado previamente, use

LOADT

pero recuerde, ésto borra lo que se encuentre allí.

CP/M

Al usar la versión CP/M, la mayor parte de lo que se acaba de explicar sobre el disco-RAM es aplicable, pero al haber sólo 11K de disco en esa versión, la única restricción para las pantallas de CP/M es el tamaño del disco. Observe que para cerciorarse que PIP puede copiar programas debe asegurarse que la página 0 está limpia, usando Ø CLEAR FLUSH al crear un nuevo fichero, bien usando un nombre nuevo al momento de cargar o usando FILE<nombre fichero>. Las partes redundantes de lo anterior son SAVET, LOADT y INIT-DISC.

El Mundo Exterior

Dos comandos están disponibles para la comunicación a través de los 'ports' normales de I/O del CPC464. Son:

n1 INP n2
n1 n2 OUTP

INP da un valor a la pila del port n1.
OUTP pone el valor n1 en el port n2.

Adiciones Varias

Los cambios finales al estándar son:

FREE

da un valor en la pila de la cantidad de almacenamiento disponible en bytes. Por ejemplo,

```
: BYTES FREE . .' bytes restantes' CR ;
```

al escribir BYTES el mensaje 18919 bytes restantes, o similar, aparecerá en pantalla.

SIZE

da un valor en la pila sobre el tamaño actual del diccionario.

f LINK

este comando, (donde f es un banderín), interconecta la pantalla y la impresora, de tal forma que lo que aparezca en pantalla será también impreso por la impresora.

En la versión CP/M hay una cantidad de comandos para manipular BDOS incorporados al sistema, la mayor parte de los cuales son para uso interno de FORTH; los únicos disponibles para el usuario son:

FILE <nombre de fichero>

Cierra el fichero .FIG actual y abre uno nuevo.

n1 n2 BDOS n3

Fija C en n2 y DE en n1 devuelve n3=A al sistema.

Capítulo 3
Características Avanzadas de FORTH

Grabando un diccionario ampliado

Como podrá ver en cualquier trabajo en FORTH, una aplicación terminada es simplemente una extensión del diccionario. En Abersoft FORTH si tiene un programa terminado y no quiere compilar de disco-RAM cada vez que quiere utilizarlo, o tiene un juego de rutinas que quiere utilizar cada vez que introduzca FORTH, éstos pueden ser grabados por el método siguiente:

Versión en cinta

1) Busque la longitud total de su programa escribiendo

```
SIZE .
```

2) Cambie los parámetros del ARRANQUE EN FRIO escribiendo lo siguiente:

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
HERE FENCE !
' FORTH 8 + 32 +ORIGIN !
```

(si usa vocabulario distinto que FORTH use 'vocab 6 + ...)

3) Escriba

```
HERE ENVIR
```

para grabar su nueva versión ampliada de FORTH en cinta. (Use su cinta propia para ésto y no la cinta master de FORTH). Esta nueva cinta es para su uso propio, naturalmente, y no para re-venderla, alquilarla o prestarla.

4) Cuando tenga un programa que no haya que cambiar o si quiere crear un programa para vender, debe cerciorarse que la aplicación no regresará al nivel de comando, por lo tanto, escriba

```
ZAP <palabra>
```

donde <palabra> es la palabra final del programa, como la palabra GAME en el ejemplo del Capítulo 2. Esto guardará una reproducción del sistema en cinta. Este nuevo programa puede ejecutarse con RUN" (pulsando las teclas CTRL y ENTER pequeña simultáneamente).

Versión CP/M

1) Busque la longitud total de páginas grabadas de su programa, escribiendo

```
HERE 256 / .
```

2) Cambie los parámetros de ARRANQUE EN FRIO escribiendo lo siguiente:

```
FORTH DEFINITIONS DECIMAL
LATEST 12 +ORIGIN !
HERE 28 +ORIGIN !
HERE 30 +ORIGIN !
HERE FENCE !
' FORTH 8 + 32 +ORIGIN !
```

(si usa un vocabulario distinto que FORTH use 'vocab 6 + . . .)

3) Escriba

```
MON
```

para salir de FORTH y después

```
SAVE n FORTHEX.COM
```
(donde n es el número encontrado en 1)

para grabar su nueva versión ampliada de FORTH en disco. (Utilice un disco propio, NO el disco master de FORTH). Este nuevo disco será para su uso propio, naturalmente, y no para re-vender, alquilar o prestar.

4) Cuando tenga un programa que no haya que cambiar o si quiere crear un programa para vender, debe cerciorarse que la aplicación no regresará al nivel de comando, por lo tanto, escriba,

```
ZAP <palabra>
```

donde <palabra> es la palabra final del programa, como la palabra GAME en el ejemplo del Capítulo 2. Esto guardará una reproducción del sistema en disco. Este nuevo programa puede ejecutarse con

```
<nombre del programa>
```

igual que cualquier otro en CP/M.

Uso de Registros

El programador que quiera usar rutinas en código-máquina (usando CREATE o ; CODE) o Ensamblador en 1.1B necesitará hacer uso de registros de Abersoft FORTH.

REGISTROS

Son los siguientes:

FORTH	Z80	
IP	BC	Debe ser conservado entre palabras.
W	DE	Entrada sólo cuando se usa PUSHDE.
SP	SP	Pila
UP	IX	Puntero de área de usuario. Debe ser conservado entre palabras.
	HL	Entrada sólo cuando se usa PUSHHL.

Para entender debidamente lo anterior requerirá el libro de Ting, como se mencionó anteriormente.

Capítulo 4
Ensamblador FORTH

Ensamblador

Permite con facilidad la construcción de palabras enteras usando:

```
CODE. . .(nemónicos de ensamblador). . .C ;
```

o nuevas palabras definidas usando

```
: . . . . ; CODE . . .(nemónicos de ensamblador). . .C ;
```

Por ejemplo:

```
CODE DOUBLE  (coger palabra en la cima de la pila y duplicarla)
HL POP       (tomar palabra de arriba de la pila)
HL DAD       (ADD HL,HL)
HL PUSH      (reponer palabra en la cima de la pila)
NEXT         (compila, salta al siguiente valor)
C;
```

Sub-rutinas que pueden ser necesarias en una palabra FORTH de bajo nivel pueden ser definidas así:

```
LABEL SUB1
HL DCX (decrementar HL)
RET
C;

CODE DECREMENT
HL POP
SUB1 CALL
PUSHHL (restaurar el valor original de HL)
C;
```

Como puede verse en el ejemplo anterior, el Ensamblador FORTH está estructurado en Notación Polaca como el resto del lenguaje. A continuación aparece el Ensamblador FORTH en su totalidad, con los nemónicos estándar de Z80 como referencia:

Donde:

r = A, B, C, D, E, H, L o (HL)
cc = condición (Z, NZ etc.)
rp = par de registros
ir = registros índices IY o IX

FORTH	Z80	FORTH	Z80	FORTH	Z80
CCF	CCF	DI	DI	XRA	XOR A
EI	EI	CPL	CPL	HALT	HALT
DAA	DAA	NOP	NOP	SCF	SCF
EXAF	EX AF,AF'	RET	RET	EXX	EXX
EXDEHL	EX DE,HL	LDADE	LD A,(DE)	LDABC	LD A,(BC)
STADE	LD (DE),A	STABC	LD (BC),A	IMn	IMn
NEG	NEG	LDAI	LD A,I	LDAR	LD A,R
RETI	RETI	RETN	RETN	LDRA	LD R,A
LDIA	LD I,A	r1 r2 MOV	LD r2,r1	n r MVI	LD r,n
n rp LXI	LD rp,nn	rp n SXR	LD (n),rp	n rp LXR	LD rp,(n)
r INC	INC r	n RST	RST n	rp POP	POP rp
rp PUSH	PUSH rp	rp INX	INC rp	r DCR	DEC r
rp DCX	DEC rp	rp DAD	ADD HL,rp	r ADD	ADD A,r
n ADI	ADD A,n	r ADC	ADC A,r	n ACI	ADC A,n
r SUB	SUB r	n SUI	SUB n	r SBC	SBC r
n SCI	SBC n	r AND	AND r	n ANI	AND n
r XOR	XOR r	n XRI	XOR n	r OR	OR r
n ORI	OR n	r CMP	CMP r	n CMPI	CMP n
n IN	IN A,(n)	n OUT	OUT A,(n)	DADX	ADD IX,IX
rp DSBC	SBC HL,rp	n JMP	JMP n	n CALL	CALL n
DADY	ADD IY,IY	rp ir DAD	ADD ir,rp	rp DADC	ADC HL,rp
n STA	LD (n),A	n LDA	LD A,(n)	EX(SP)	EX(SP),HL
ir!HL JPI	JP (HL)!(ir)	SPHL	LD SP,HL	r RLC	RLC r
r RL	RL r	r RRC	RRC r	r RR	RR r
r SLA	SLA r	r SRL	SRL r	r SRA	SRA r
RRCA	RRCA	RLA	RLA	RRA	RRA
r n BIT	BIT n,r	r n RES	RES n,r	r n SET	SET n,r
LDI	LDI	CPI	CPI	INI	INI
OUTI	OUTI	LDD	LDD	CPD	CPD
IND	IND	OUTD	OUTD	LDIR	LDIR
CPIR	CPIR	INIR	INIR	OTIR	OTIR
LDDR	LDDR	CPDR	CPDR	INDR	INDR
OTDR	OTDR	r IN(C)	IN r,(c)	r OUT(C)	OUT r(c)
n cc CALLC	CALL cc,n	n cc RETC	RET cc,n	n cc JPC	JP cc,n
n cc JRC	JR cc,n	n JR	JR n	n DJNZ	DJNZ n

Los saltos y bucles pueden ser empleados como en lenguaje ensamblador normal, pero puede ser difícil. El Ensamblador FORTH permite estructuras de bucles similares a las del FORTH normal.

Estas son:

```
BEGIN. . .AGAIN (bucle infinito)

BEGIN. . .cc UNTIL
```

éste es el equivalente de:

```
L1: <nemonimos>
          JR cc,L1 (o si es salto >128 ,usa JP cc,L1)
BEGIN. . .cc WHILE. . .REPEAT
cc IF. . .ENDIF
cc IF. . .ELSE. . .ENDIF
```

Hay también un DO. . .LOOP que usa la instrucción DJNZ, por ejemplo:

```
5 DO A INR LOOP
```

Se ensamblaría:

```
     LD B,5
L1:  INC A
     DJNZ L1
```

Puesto que los nombres de registros A, B, C, etc., se confunden fácilmente con los números hexadecimales, y también que la mayoría de las personas prefieren escribir ensamblador en HEX, recuerde especificar los números anteponiendo un (cero) Ø por ejemplo: ØA, ØB, etc.

El Ensamblador FORTH no verifica del todo la forma en que se escribe el código, por lo que puede fácilmente permitir instrucciones ilegales. Verifica al final del ensamblado que la pila esté vacia, pero la responsabilidad de revisar los errores es SUYA.

Para pasar valores a la pila, y retornar a la siguiente palabra FORTH, los registros mencionados deben ser preservados. Para regresar sin ningún valor, NEXT ensambla un salto al FORTH NEXT; es decir, use NEXT y no NEXT JMP al final del ensamblado. Si tiene un valor páselo en el registro HL y use PUSHHL. Si tiene dos, páselos en los registros HL y DE y use PUSHDE. Esto dejará la pila así:

DE HL (tos).

Capítulo 5
El Editor de Disco

Preparación del Disco-RAM

FORTH organiza el almacenamiento masivo como pantallas de 1024 caracteres. El disco-RAM tiene 11K y sus pantallas están numeradas del 0 al 10.

Cada pantalla está organizada por el sistema en 16 líneas de 64 caracteres cada una. Las pantallas de FORTH son un arreglo de la memoria virtual y no se corresponden con el formato de pantalla del CPC464.

Accediendo a una Pantalla

Para comenzar una sesión de edición, escriba EDITOR. Esto llama al vocabulario EDITOR y permite que se introduzca texto. Usa INIT-DISC al comenzar para limpiar el disco-RAM.

Se selecciona la pantalla a editar, usando bien:

n LIST (lista la pantalla n y selecciona para edición) o

n CLEAR (limpia la pantalla n y selecciona para edición)

Para escribir texto después de usar LIST o CLEAR, se utiliza el comando P (put).

Ejemplo:

```
0 P   Así es cómo
1 P   se introduce texto
2 P   en las líneas 0, 1 y 2 de la pantalla elegida.
```

Edición de Líneas

Durante esta descripción del editor se hace referencia a PAD. Este es el buffer de texto. Puede contener una línea de texto usada o grabada por un comando de edición de línea, o una cadena de texto, que buscar o borrar por un comando de edición de cadena.

PAD puede ser usado para transferir una línea de una pantalla a otra, así como realizar operaciones de edición dentro de una pantalla.

Comandos de Edición de Líneas:

- n H — Mantener línea n en PAD (usado por el sistema con más frecuencia que por el usuario).
- n D — Borrar línea n pero mantenerla en PAD. La línea 15 se borra al moverse las líneas, n+1 a la 15, una línea hacia arriba.
- n T — Escribir línea n y guardarla en PAD.
- n I — Insertar el texto en la línea n desde PAD, moviendo la anterior línea n y siguientes, hacia abajo. La línea 15 se pierde.
- n E — Borrar la línea n con espacios en blanco
- n S — Desplazar línea n. Las líneas n y subsiguientes se mueven hacia abajo una línea. La n se convierte en blanco. La 15 se pierde.

Edición de Cadenas

La pantalla de texto en edición reside en un buffer de almacenamiento. El cursor de edición es una variable con un desplazamiento en esta zona. Se proveen comandos para que el usuario posicione el cursor, bien directamente o buscando una cadena de texto, y para insertar o borrar texto en la posición del cursor.

Comandos para posicionar el cursor.

TOP	Sitúa el cursor al comienzo de la pantalla.
n M	Mueve el cursor una cantidad n, con signo, e imprime el cursor.
n LIST	Lista la pantalla n y la selecciona para edición.
n CLEAR	Limpia la pantalla n con blancos y la selecciona para edición.
n1 n2 COPY	Copia la pantalla n1 en la pantalla n2.
L	Lista la pantalla actual. Se vuelve a listar la línea del cursor, después que se ha listado la pantalla, para mostrar su posición.
F <texto>	Busca hacia adelante, desde la posición actual del cursor, hasta que encuentra la cadena <texto>. El cursor se queda al final de la cadena y se imprime la línea del cursor. Si no encontrara la cadena se da un mensaje de error y el cursor se coloca en la aprte superior de la pantalla.
B	Usada después de F para hacer retroceder al cursor toda la longitud del texto más reciente.
N	Buscar la siguiente ocurrencia de la cadena encontrada por un comando F.
X <texto>	Buscar y borrar la cadena <texto>.
C <texto>	Copiar texto en la línea del cursor, en la posición del cursor.
TILL <texto>	Borrar en la línea del cursor, desde el cursor hasta el final de la cadena de texto <texto>.

NOTA:

Teclear C sin texto colocará un comando nulo en el texto, en la posición del cursor. Esto detendrá abruptamente, más adelante, la compilación. Para borrar esta clase de error escriba TOP X **ENTER**.

El Editor de Pantalla

Accediendo a una Pantalla

Para comenzar una sesión de edición, escriba SCREDIT. Esto llama al vocabulario del EDITOR DE PANTALLA y permite introducir textos. Igual que en la edición de líneas, antes de empezar utiliza INIT-DISC para limpiar el disco-RAM.

La pantalla a editar se selecciona usando:

n SCRED (lista la pantalla n y la selecciona para edición).

El editor comienza en modo de SOBREESCRITURA. El cursor puede ser desplazado con las teclas de flechas y cualquier cosa que se teclee será mostrada en la pantalla. ENTER simplemente mueve el cursor hasta el comienzo de la siguiente línea. DEL borra el carácter anterior como es de esperar. Debe tenerse cuidado al final de las líneas de la pantalla, ya que FORTH las considera como un bloque único, contínuo; por lo que, si una palabra finaliza en la columna 64, considerará a la columna 1 de la siguiente línea como parte de la anterior. El otro modo es el de INSERCION; el cambio entre los modos se logra con COPY. En este modo todos los caracteres escritos desplazan a los caracteres de la línea en la que se encuentra, perdiendo el exceso en el extremo derecho. DEL cierra una línea, y ENTER realiza la misma función que en modo de SOBREESCRITURA. Otros dos comandos, CTRL S y CTRL D, o bien abre un espacio en la pantalla, en la posición actual, del cursor perdiendo la línea 15, o borra la línea actual, llenando la decimoquinta con espacios. Para completar la edición, se pulsa ESC. Se ofrece, a continuación, la opción de completar la edición, o de interrumpirla, dejando la pantalla como estaba.

Capítulo 6
Mensajes de Error

Si el compilador encuentra un error, donde sea, limpia el dato y la pila, dando un mensaje de error con un valor numérico. El significado de estos mensajes de error son:

MENSAJE	Significado
0	No se encuentra la palabra
1	Pila vacia
2	Diccionario lleno
3	Modo de dirección incorrecto
4	No es único
6	¿Límites del Disco-RAM? (pág. no del 0 al 10)
7	Pila llena
9	Tratando de cargar desde la pág. 0
17	Compilado solamente, use en definición
18	Ejecución solamente
19	Condicionales no por pares
20	Definición no terminada
21	En diccionario protegido
22	Use sólo al cargar
23	Fuera de la actual pantalla de edición.
24	Declare vocabulario.

Capítulo 7
Guía de Referencia FORTH

La guía de referencia contiene todas las definiciones de palabras en esta edición de Fig-FORTH (las extensiones para el Amstrad CPC464 han sido presentadas en capítulos anteriores) en el vocabulario principal. Son presentadas en orden ASCII.

La primera línea muestra una descripción simbólica de la acción de proceso en la pila de parámetros. Los símbolos indican el orden en el que los parámetros de entrada han sido colocados en la pila. Tres guiones '---' indican el punto de ejecución; cualquier parámetro que quede en la pila es listado. En esta notación, la cima de pila está a la derecha.

Los símbolos incluyen:

addr	dirección de memoria
b	byte de 8 bits (v.g. 8 bits altos, cero)
c	carácter ASCII de 7 bits
d	doble entero con signo de 32 bits
f	banderín de Boole. 0=falso, distinto de cero=verdadero
ff	banderín de Boole, falso = 0
n	número entero con signo de 16 bits
u	número entero sin signo de 16 bits
tf	banderín de Boole, verdadero = distinto de cero

A menos que se especifique lo contrario, toda referencia a los números serán de enteros con signo de 16 bits. El byte alto se encuentra en la cima de la pila, con el signo en el bit más a la izquierda (más significativo). Para números de 32 bits la parte más significativa se encuentra arriba.

Toda la aritmética, implícitamente, es de enteros con signo de 16 bits, sin especificar errores o si ha habido overflow (rebose).

Expresamos nuestro reconocimiento a FORTH INTEREST GROUP por algunas partes de este compilador y del manual, recomendándose hacerse miembro de este augusto cuerpo. La dirección de Fig (Reino Unido) es.

The Membership Secretary
FIG(UK)
24 Western Avenue
Woodley
Reading
RG5 3BH

! **n addr —**

Guarda 16 bits de n en la dirección. Se pronuncia 'store'.

!CSP

Guarda la posición de la pila en CSP. Usado como parte de la seguridad del compilador.

**d1 — d2**

Genera de un número doble d1, el siguiente carácter ASCII, el que se colocará en una cadena de salida. El resultado d2 es el cociente después de dividir entre BASE y se mantiene para su procesamiento posterior. Usado entre < # y #>. Vea # S.

#> **d — addr count**

Termina la conversión de salida numérica deshechando d, dejando la dirección del resto y el contador de caracteres disponible para TYPE.

#BUF **—n**

Una constante que da la cantidad de buffers de disco asignados.

#S **d1 — d2**

Genera texto ASCII en el buffer de salida de texto, por el uso de #, hasta que de como resultado un cero en doble número. Usado entre < # y #>.

' **— addr**

Usado en la forma: ·

' nnnn

Deja la dirección de campo del parámetro de la palabra del diccionario nnnn. Como directivo del compilador, ejecuta en una definición de dos puntos (:) para compilar la dirección como literal. Si no se encuentra la palabra después de una búsqueda de CONTEXT y CURRENT, se da un mensaje apropiado de error. Se pronuncia 'tick'.

(

Usado en la forma:

(cccc)

Ignora un comentario que estará delimitado por un paréntesis derecho en la misma línea. Puede ocurrir durante la ejecución o en una definición de dos puntos (:). Se requiere un espacio en blanco después de abrir paréntesis.

(.")

Procedimiento de ejecución, compilado por .", y que transmite el texto que le sigue al periférico seleccionado. Vea ."

(;CODE)

El procedimiento de ejecución compilado por ;CODE que re-escribe el campo del código de la última palabra definida para que apunte a la siguiente secuencia de código máquina. Vea ;CODE.

(+LOOP) n —

El procedimiento de ejecución compilado por +LOOP, que incrementa el índice del bucle en n y comprueba si éste ha terminado. Vea +LOOP.

(ABORT)

Se ejecuta después de un error cuando WARNING es -1. Esta palabra normalmente ejecuta ABORT, pero puede ser modificada (con cuidado) para un procedimiento alternativo del usuario.

(DO)

El procedimiento de ejecución compilado por DO que desplaza los parámetros de control de bucle a la pila de retorno. Vea DO.

(FIND) **addr1 addr2 — pfa b tf (ok)**
addr1 addr2 — ff (bad)

Busca en el diccionario, comenzando por el nombre cuya dirección es addr2, hasta que éste coincide con el de la dirección addr1. Da la dirección, longitud en bytes y verdadero (según Boole) si ha coincidido la comparación. Si no hay coincidencia, la respuesta es falso.

(LINE) **n1 n2 — addr count**

Convierte la línea n1 y la pantalla n2 a la dirección del buffer de disco que contiene los datos. Un 40 indica la longitud total de la línea de texto.

(LOOP)

El procedimiento de ejecución compilado por LOOP que incrementa el índice del bucle y prueba si éste ha terminado. Vea LOOP.

(NUMBER) **d1 addr1 — d2 addr2**

Convierte el texto ASCII que comienza en la dirección addr1+1, con relación a BASE. El nuevo valor se acumula en el número doble d1, siendo convertido en d2. Addr2 es la dirección del primer dígito no convertible. Usado por NUMBER.

* **n1 n2 — prod**

Coloca el producto, con signo, de la multiplicación de dos números.

*/ **n1 n2 n3 — n4**

Hace la razón n4 = n1*n2/n3, donde todos los números son con signo. La retención de un producto intermedio de 31 bits permite mayor precisión que la disponible con la secuencia: n1 n2 * n3 /

***/MOD** **n1 n2 n3 — n4 n5**

Coloca el cociente en n5 y el resto en n4, de la operación n1 * n2 / n3. Se utiliza un producto intermedio de 31 bits como para * /.

+ **n1 n2 — sum**

Coloca la suma de n1 + n2

+! **n addr —**

Suma n al valor en la dirección especificada. Se pronuncia 'plus-store'.

+- **n1 n2 — n3**

Se aplica el signo de n2 a n1, y se coloca en n3

+BUF **addr1 — addr2 f**

Avanza la dirección addr1 del buffer de disco a la dirección del siguiente buffer addr2. La f es el falso booleano cuando addr2 es el buffer al que apunta la variable PREV.

+LOOP **n1 — (run)**
addr n2 — (compila)

Usada en una definición de dos puntos (:) en la forma:

DO . . . n1 +LOOP

Al momento de ejecutarse +LOOP controla, por selección, el regreso al correspondiente DO, basado en n1, el índice del bucle y su límite.

El incremento con signo n1 se añade al índice y el total se compara con el límite. El regreso a DO tiene lugar hasta que el nuevo índice sea igual a o mayor que el límite (n1 0). Al salir del bucle se desechan los parámetros y la ejecución sigue adelante.

Al momento de la compilación +LOOP compila la palabra (+LOOP) y el desplazamiento del salto calculado desde HERE hasta la dirección que ha dejado en la pila Do. n2 se usa para verificación de errores del tiempo de compilación.

+ORIGIN n — addr

Coloca la dirección de memoria relativa a n, del área de origen del parámetro. n es la unidad de dirección mínima, sea byte o palabra. Esta definición es usada para acceder o modificar los parámetros en el área de origen.

.CPU

Imprime el mensaje al encender su ordenador.

' n —

Guarda n en la siguiente celda de memoria disponible del diccionario, adelantando el puntero del diccionario. (coma)

- n1 n2 — diff

Da la diferencia de n1 - n2

—>

Continúa la interpretación con la siguiente pantalla de disco. Se llama 'next-screen' -siguiente pantalla-.

-DUP n1 — n1 (si es cero)
n1 -- n1 n1 (si no es cero)

reproduce n1 sólo si no es cero. Normalmente se utiliza para copiar un valor antes de IF, para eliminar la necesidad de un ELSE que lo deseche.

-FIND — pfa b tf (encontrado)
— ff (no encontrado)

Acepta la siguiente palabra de texto (delimitada por espacios en blanco) en la entrada a HERE y busca en CONTEXT y después en los vocabularios de CURRENT para comparar. Si encuentra una palabra dejará la dirección del parámetro en el diccionario, su longitud en bytes y un 'verdarero' (Boole). De lo contrario, quedará sólo un 'falso'.

-TRAILING **addr n1 — addr n2**

Ajusta el contador de caracteres n1 en la dirección de comienzo de una cadena de texto para suprimir la salida de caracteres en blanco

. **n —**

Imprime un número con signo, de 16 bits en complemento a dos, convertido según la BASE numérica. Le sigue un espacio en blanco. Se llama 'dot' (punto).

."

Usado en la forma:

." cccc"

Compila una cadena de caracteres cccc (delimitada por las comillas finales), con un procedimiento de ejecución para trasmitir texto al periférico seleccionado. Si se ejecuta fuera de una definición, ." imprimirá el texto hasta las comillas finales. Vea (.")

.LINE **line scr —**

Imprime en pantalla una línea de texto del disco RAM por sus números de línea y pantalla. Se suprimen los espacios en blanco que puedan seguir,

.R **n1 n2 —**

Imprime el número n1, alineado a la derecha, de un campo cuya anchura es n2. No se imprimen los espacios en blanco que sigan.

/ **n1 n2 — quot**

Coloca el cociente con signo de n1 / n2.

/MOD **n1 n2 — rem quot**

Coloca el resto y el cociente con signo de n1 / n2. El resto tendrá el signo del dividendo.

0123 **— n**

Estos números pequeños se usan tan frecuentemente que es interesante definirlos por nombre en el diccionario como constantes.

0< **n — f**

Coloca un banderín de 'verdadero' si el número es menor que cero (negativo), de lo contrario dejará un banderín de 'falso'.

0= **n — f**

Coloca un banderín de verdadero si el número es igual a cero, de lo contrario dejará un banderín de 'falso'.

0BRANCH f —

El procedimiento de ejecución para saltos condicionales. Si f es falso el parámetro siguiente se suma al puntero para bifurcar hacia adelante o hacia atrás. Compilado por IF, UNTIL y WHILE.

1+ **n1 — n2**

Incrementa n1 en 1.

2+

Incrementa n1 en 2.

2! **nlow nhigh addr —**

Almacenamiento de 32 bits. nhigh se almacena en la dirección addr; nlow en addr+2.

2 @ **addr — nlow nhigh**

Cargar 32 bits. El contenido de nhigh se carga de addr; nlow de addr+2.

2CONSTANT

Una palabra definible usada de la forma:

```
d 2CONSTANT cccc
```

para crear la palabra cccc, con su parámetro conteniendo d. Al ejecutarla más adelante, lleva el valor doble de d a la pila.

2DROP **d —**

Desechar el número doble de la pila.

2DUP **n2 n1 — n2 n1 n2 n1**

Duplica los dos valores en la cima de la pila. Equivale a OVER OVER

2OVER **d1 d2 — d1 d2 d1**

Copia el primer valor doble d1 a la cima de la pila.

2SWAP **d1 d2 — d2 d1**

Intercambia los dos primeros valores dobles de la pila.

2VARIABLE

Una palabra definible usada de la forma:

```
d VARIABLE cccc
```

Al ejecutar VARIABLE crea la definición cccc con su campo de parámetro inicializado a d. Al ejecutar cccc después, la dirección del parámetro (que contiene d) queda en la pila, pudiendo acceder a ella con una doble búsqueda o doble almacenamiento.

:

Usado en la forma llamada definición de dos puntos

```
: cccc ... ;
```

Crea una palabra cccc en el diccionario, definiéndola como equivalente a la secuencia de definiciones FORTH que le siguen '...' hasta el siguiente ; o ;CODE. El proceso de compilación lo realiza el intérprete de texto mientras que STATE no sea cero. Otros detalles son que el vocabulario de CONTEXT se iguala al de CURRENT y que las palabras con el bit de precedencia (P) distinto de cero son ejecutadas en vez de ser compiladas.

;

Finaliza una definición de dos puntos y detiene cualquier compilación posterior. Compila el tiempo de ejecución ;S.

;CODE

Usado en la forma

```
: cccc ... ; CODE
```

nemónicos del ensamblador

Detiene la compilación y finaliza la definición de una nueva palabra cccc al compilarla (;CODE). Pone el vocabulario CONTEXT en ASSEMBLER (ensamblador), ensamblando en lenguaje máquina los nemónicos que siguen. Si no se carga el ASSEMBLER, los valores de los códigos pueden ser compilados usando , y C,.

Cuando cccc se ejecuta después de la forma:

```
cccc nnnn
```

la palabra nnnn será creada con su procedimiento de ejecución dado por el código máquina que sigue a cccc. Es entonces cuando se ejecuta nnnn, y lo hace saltando hasta el código que sigue a nnnn. Una palabra definida debe existir en cccc antes de ;CODE.

;S

Detiene la interpretación de una pantalla. ;S es también la palabra de tiempo de ejecución compilada al final de una definición de dos puntos, que regresa la ejecución al procedimiento de llamada (call).

< n1 n2 — f

Pone un banderín de verdadero si n1 es menor que n2; de lo contrario pone uno de falso.

< #

Utilizado para el formato numérico de salida con las palabras:

```
< # # # S SIGN # >
```

La conversión se lleva a cabo en un número doble que produce texto en PAD.

<BUILDS

Usado en una definición de dos puntos

```
: cccc <BUILDS ...
DOES> ... ;
```

Cada vez que se ejecuta cccc, < BUILDS define una palabra nueva con un procedimiento de ejecución de alto nivel. Ejecutando cccc en la forma:

```
cccc nnnn
```

utiliza < BUILDS para crear una palabra en el diccionario para nnnn con una llamada a la parte DOES > para nnnn. Al ejecutar nnnn después, ésta tiene la dirección de su área de parámetro en la pila y ejecuta las palabras que siguen a DOES > en cccc. < BUILDS y DOES > permiten que los procedimientos de tiempo de ejecución sean escritos en alto nivel y no en código máquina (como se requiere con ;CODE).

= **. n1 n2 — f**

Pone un banderín de verdadero si n1=n2; de lo contrario pone uno falso.

> **n1 n2 — f**

Pone un banderín de verdadero si n1 es mayor que n2; de lo contrario pone uno de falso.

>R **n —**

Quita un número de la pila de cálculos y lo coloca en la posición más accesible de la pila de retorno. Su uso debe estar equilibrado con R > en la misma definición.

? **addr —**

Imprime el valor contenido en la dirección, en formato libre según la base vigente.

?COMP

Da mensaje de error si no está compilando.

?CSP

Da mensaje de error si la posición de la pila difiere del valor guardado en CSP.

?ERROR **f n —**

Da mensaje de error, número n, si el banderín es verdadero.

?EXEC

Da mensaje de error si no está ejecutando.

?LOADING

Da mensaje de error si no está cargando.

?PAIRS **n1 n2 —**

Da mensaje de error si n1 no es igual a n2. El mensaje indica que las condicionales compiladas no concuerdan.

?STACK

Da mensaje de error si la pila está fuera de límites.

?TERMINAL **— f**

Realiza una prueba de la tecla 'break' en el teclado del terminal. Un banderín de verdadero indica que ha sido pulsada.

@ **addr — n**

Pone el contenido, de 16 bits, de la dirección.

ABORT

Limpia las pilas y entra en estado de ejecución. Devuelve el control a los operadores del terminal, imprimiendo un mensaje apropiado a la instalación.

ABS **n — u**

Pone el valor absoluto de n en u.

AGAIN **addr n — (compilando)**

Usado en una definición de dos puntos, en la forma

```
BEGIN. . .AGAIN
```

Al momento de ejecutarse, AGAIN obliga el regreso de la ejecucón al correspondiente BEGIN. No hay efecto en la pila. La ejecución no puede salir de este bucle (a menos que R DROP se ejecute un nivel más abajo).

Al momento de la compilación, AGAIN compila a BRACH con una compensación desde HERE a addr. n se utiliza para verificar errores al momento de compilar.

ALLOT **n —**

Suma el número con signo al puntero del diccionario DP. Puede usarse para reservar espacio en el diccionario o re-localizar el origen de memoria. n se refiere al tipo de dirección de memoria (bien sea byte o palabra).

AND **n1 n2 — n3**

Pone el bit lógico 'and' de n1 y n2 en n3.

ASSEMBLER

El vocabulario que guarda el ensamblador FORTH.

B/BUF **— n**

Esta constante muestra la cantidad de bytes por buffer de disco; la cantidad de bytes leidos del disco por BLOCK.

B/SCR **— n**

Esta constante muestra la cantidad de bloques por pantalla de edición. Por convención, una pantalla tiene 1024 bytes organizados en 16 líneas de 64 caracteres.

BACK **addr —**

Calcular la diferencia de la bifurcación hacia atrás entre HERE y addr, y compila en la siguiente dirección de memoria disponible del diccionario.

BASE **— addr**

Una variable definida por el usuario que contiene el número base vigente, utilizado para la conversión de input/output.

BEGIN **— addr n (compilando)**

Tiene lugar en una definición de dos puntos, en la forma

```
BEGIN ... UNTIL
BEGIN ... AGAIN
BEGIN ... WHILE ... REPEAT
```

Al momento de ejecutarse, BEGIN marca el comienzo de una secuencia que puede ser ejecutada repetitivamente. Sirve como punto de retorno del correspondiente UNTIL, AGAIN o REPEAT. Al ejecutarse UNTIL, tendrá lugar un regreso a BEGIN si la cima de la pila es falso; con AGAIN y REPEAT siempre tiene lugar un regreso a BEGIN. Al momento de la compilación BEGIN deja su dirección de regreso y n para verificar errores.

BL **— c**

Una constante que pone los valores ASCII de 'espacio en blanco'.

BLANKS **addr count —**

Llena con espacios en blanco un área de la memoria que comienza en addr.

BLK **— addr**

Una variable definida por el usuario conteniendo el número del bloque que está siendo interpretado. Si es cero, toma el valor del buffer de entrada del terminal.

BLOCK **n — addr**

Pone la dirección de memoria del buffer de bloques que contiene al bloque n. Si el bloque no estuviera ya en memoria, se transferirá del disco al buffer que se hubiera escrito el último. Si el bloque que ocupa ese buffer está señalado como actualizado, éste se vuelve a escribir en el disco antes que el bloque n sea colocado en el buffer. Vea también BUFFER, R/W, UPDATE, FLUSH.

BRANCH

Procedimiento del momento de ejecución para bifurcar incondicionalmente. Un desplazamiento en-línea se suma al puntero intérprete IP para saltar hacia atrás o hacia adelante. BRANCH es compilado por ELSE, AGAIN, REPEAT.

BUFFER **n — addr**

Buscar el siguiente buffer de memoria y asignarlo al bloque n. Si el contenido del buffer está señalado como actualizado será escrito en el disco. El bloque no es leído del disco. La dirección será la primera celda dentro del buffer para el almacenamiento de datos.

C! **b addr —**

Guarda 8 bits en la dirección.

C/L **— n**

Constante del número de caracteres por línea; usado por el editor.

C, **b —**

Guarda 8 bits de b en el siguiente byte disponible del diccionario, adelantando el puntero del diccionario.

C;

Finaliza una definición en ensamblador.

C@ **addr — b**

Pone el contenido, en 8 bits, de la dirección de memoria.

CASE — n (compilando)

Tiene lugar en una definición de dos puntos, en la forma

```
CASE
n OF ... ENDOF
...
ENDCASE
```

Al momento de ejecutarse, CASE marca el comienzo de una secuencia de comandos OF ... ENDOF. Al momento de compilar, CASE asigna n para verificar errores.

CFA pfa — cfa

Convierte la dirección de campo del parámetro de una definición en la dirección de campo del código.

CLS

Realiza la función de limpiar la pantalla y devuelve el cursor al origen.

CMOVE from to count —

Desplaza la cantidad especificada (count) de bytes desde la dirección from hasta la dirección to. El contenido de la dirección from se desplaza primero, dirigiéndose hacia la memoria alta.

CODE

Crea una nueva palabra, y pone el vocabulario vigente en ASSEMBLER (ensamblador), a punto para la definición de códigos.

COLD

El procedimiento de arranque en frío para ajustar el puntero del diccionario al estándar mínimo y re-comenzar vía ABORT. Puede ser llamado desde el terminal para quitar programas de aplicaciones y volver a empezar.

COMPILE

Al ejecutarse la palabra que contiene COMPILE, la dirección de ejecución de la palabra que sigue a COMPILE es copiada (compilada) en el diccionario. Esto permite que puedan ser manejadas situaciones específicas de compilación además de compilar simplemente una dirección de ejecución (lo que ya hace el intérprete).

CONSTANT n —

Una palabra de definición usada en la forma

n CONSTANT cccc

para crear la palabra cccc, con su campo de parámetro conteniendo n. Al ejecutarse cccc más adelante, pondrá (push) el valor de n en la pila.

CONTEXT — addr

Una variable definible por el usuario conteniendo un puntero del vocabulario, por dentro del cual las búsquedas del diccionario comenzarán primero.

COUNT addr1 —— addr2 n

Pone la dirección del byte addr2 y la cantidad de bytes n de un texto de mensaje que comienza en la dirección addr1. Se supone que el primer byte en la dirección addr1 contiene la cantidad de bytes del texto y que éste comienza en el segundo byte. Normalmente, a COUNT le sigue TYPE.

CR

Transmite un retorno de carro y salto de línea al periférico seleccionado.

CREATE

Una palabra de definición usada en la forma

```
CREATE cccc
```

por palabras tales como CODE y CONSTANT para crear una cabecera del diccionario para una definición FORTH. El campo del código contiene la dirección del campo del parámetro de las palabras. La nueva palabra es creada en el vocabulario CURRENT.

CSP **— addr**

Una variable definible por el usuario que almacena temporalmente la posición del puntero de la pila, para verificación de errores de compilación.

D+ **d1 d2 — dsum**

Pone la suma en número doble de dos números dobles.

D+ **-d1 n — d2**

Le pone el signo de n al número doble d1, y coloca el resultado en d2.

D. **d —**

Imprime un número doble con signo de 32 bits en complemento a dos. Los 16 bits más altos son más accesibles en la pila. La conversión se realiza de acuerdo con la BASE vigente. Le sigue un espacio en blanco. Se llama 'D-punto'.

D.R **d n —**

Imprime un número doble con signo d alineado a la derecha en un campo de n caracteres de ancho.

DABS **d — ud**

Pone el valor absoluto ud de un número doble.

DECIMAL

Fija la BASE de conversión numérica para salida/entrada en decimal (input/output).

DEFINITIONS

Usado en la forma

```
cccc DEFINITIONS
```

Pone el vocabularo de CONTEXT en el vocabulario CURRENT. En el ejemplo, a ejecutar el nombre de vocabulario cccc lo incluyó en el vocabulario CONTEXT y al ejecutar DEFINITIONS hizo que ambos especificaran el vocabulario cccc.

DIGIT **c n1 — n1 tf (ok)**
c n1 — ff (mal)

Convierte el carácter ASCII c (usando la base n1) en su equivalente binario n2, acompañado de un banderín de verdadero. Si la conversión no es válida, pone sólo un banderín de falso.

DLITERAL d — d (al ejecutar)
d — d (al compilar)

Al compilar, compila un número doble de la pila en un literal. La ejecución posterior de la definición que contiene al literal lo colocará (push) en la pila. En la ejecución, el número permanecerá en la pila.

DMINUS d1 — d2

Convierte d1 en un número doble en complemento a dos.

DO **n1 n2 — (al ejecutar)**
addr n — (al compilar)

Tiene lugar en una definición de dos puntos de la forma

DO ... LOOP DO ... +LOOP. En el momento de ejecución DO comienza una secuencia de ejecución repetitiva controlada por un límite de bucle n1 y un índice con valor inicial n2. DO obtiene éstos de la pila. Al llegar a LOOP el índice se incrementa en 1. Hasta que el índice igual o exceda a límite, la ejecución regresa a lo que sigue a DO; de lo contrario los parámeros del bucle son ignorados y la ejecución continúa adelante. n1 y n2 se determinan en el momento de ser ejecutados y pueden ser el resultado de otras operaciones. Una I dentro de un bucle pondrá el valor actual del índice en la pila. Vea I, LOOP, + LOOP, LEAVE.

Al compilar dentro de una definición de dos puntos, DO compila a (DO), dejando la siguiente dirección addr y n para verificación de error posterior.

DOES >

Una palabra que define la acción al momento de ejecutar una palabra de definición de alto nivel. DOES > modifica el campo del código y el primer parámetro de la nueva palabra para ejecutar la secuencia de direcciones de palabras compiladas que le siguen. Usada en combinación con <BUILDS. Al ejecutar, DOES > comienza con la dirección del primer parámetro de la palabra nueva en la pila. Esto permite la interpretación usando este área o su contenido. Los usos típicos incluyen ensamblador FORTH, matrices multi-dimensionales y la generación del compilador.

DP — addr

Variable definida por el usuario, puntero del diccionario, que contiene la dirección de la siguiente memoria libre por encima del diccionario. El valor puede ser leído por HERE y modificado por ALLOT.

DPL — addr

Variable definible por el usuario conteniendo la cantidad de dígitos a la derecha del punto decimal en entrada de entero doble. Puede utilizarse también para contener la localización de salida de columna de un punto decimal, en formato generado por el usuario. El valor por defecto en entrada de número sencillo es -1.

DROP n —

Elimina un número de la pila.

DUP n — n n

Duplicar el valor en la pila.

ELSE addr1 n1 — addr2 n2 (compilando)

Ocurre dentro de una definición de dos puntos en la forma:

```
IF ... ELSE ... ENDIF
```

En el momento de ejecución ELSE se ejecuta después de la parte verdadera que sigue a IF. ELSE obliga a pasar por alto la ejecución de la parte falsa que le sigue y reanuda la ejecución después del ENDIF. No tiene efecto en la pila.

Al momento de compilar, ELSE utiliza a BRANCH, reservando el desplazamiento hasta la bifurcación: y deja la dirección addr2 y n2 para verificación de errores. ELSE resuelve la bifurcación hacia adelante desde IF, al calcular la diferencia entre addr1 y HERE y almacenándola en addr1.

EMIT c —

Trasmite el carácter ASCII c al periférico de salida seleccionado. OUT se incrementa por cada salida (output) de un carácter.

EMPTY-BUFFERS

Marca todos los buffers de bloque como vacíos, sin afectar -necesariamente-a su contenido. Los bloques actualizados no son escritos en el disco. Es también un procedimiento de inicialización antes de usar el disco por primera vez.

ENCLOSE addr1 c — addr1 n1 n2 n3

El buscador de texto usado por WORD. De la dirección del texto addr1 y un carácter límite c en ASCII, se determina la diferencia en bytes hasta el primer carácter no límite n1; la diferencia hasta el primer límite después del texto n2; y la diferencia hasta el primer carácter no incluido. Este procedimiento no procesará más allá de un carácter 'nulo' ASCII, considerando éste como un delimitador incondicional.

END

Es una 'alias' o definición duplicada de UNTIL.

ENDCASE addr n — (compilar)

Ocurre en una definición de dos puntos en la forma

```
CASE n OF ... ENDOF ... ENDCASE
```

En el momento de ejecución ENDCASE marca la terminación de un comando CASE.

En el momento de compilar, ENDCASE calcula los desplazamientos hacia adelante de la bifurcación.

ENDIF **addr n — (compila)**

Ocurre en una definición de dos puntos en la forma

```
IF ... ENDIF
IF ... ELSE ... ENDIF
```

En el momento de ejecución ENDIF sirve sólo como destino de un desplazamiento hacia adelante desde IF o ELSE. Marca la terminación de la estructura condicional. THEN es otro nombre de ENDIF. Ambos nombres existen en FIG-FORTH. Vea también IF y ELSE.

En el momento de compilar, ENDIF calcula el salto adelante desde addr hasta HERE y lo guarda en addr. n se utiliza para comprobación de errores.

ENDOF **addr n — (compilar)**

Usado como ENDIF excepto en comandos CASE.

ERASE **addr n —**

Coloca en una región de memoria ceros desde la dirección addr sobre n direcciones.

ERROR **line — in blk**

Ejecuta avisos de error y reinicio de sistemas. Se examina WARNING primero. Si WARNING=positivo, n se imprime sólo como mensaje de error (instalación de RAM-disc). Si WARNING es -1, la definición (ABORT) se ejecuta, que a su vez, ejecuta el sistema ABORT. El usuario puede -con cautela- modificar esta ejecución modificando (ABORT). FIG-FORTH guarda el contenido de IN y BLK para ayudar en la localización del error. La acción final es la ejecución de QUIT.

EXECUTE **addr —**

Ejecuta la definición cuya dirección del campo del código está en la pila. La dirección del campo de código también se llama la dirección de compilación.

EXPECT **addr count —**

Transfiere caracteres del terminal a la dirección, hasta que se teclea ENTER o reciba la cantidad de caracteres. Uno o más nulos se añaden al final del texto.

FENCE **— addr**

Una variable definible por el usuario conteniendo una dirección debajo de la cual FORGET no actúa. Para que FORGET actúe más allá de este punto, el usuario debe modificar el contenido de FENCE.

FILL **addr quan b —**

Llena la memoria en la dirección con la cantidad especificada de bytes b.

FIRST **— n**

Una constante que pone la dirección del primer (más bajo) buffer de bloque.

FLD **— addr**

Una variable de usuario para el control de la anchura de campos en la salida de números. Actualmente sin uso en FIG-FORTH.

FLUSH

Escribe todos los buffers de disco actualizados en el disco-RAM.

FORGET

Ejecutado en la forma

```
FORGET cccc
```

Borra la definición llamada cccc, del diccionario con todas las entradas que le siguen físicamente. En FIG-FORTH aparecerá un mensaje de error si los vocabularios CURRENT y CONTEXT no son, en ese momento, iguales.

FORTH

El nombre del vocabulario primario. La ejecución hace a FORTH el vocabulario CONTEXT. En tanto no se definan vocabularios adicionales del usuario, las definiciones del usuario forman parte de FORTH. FORTH es inmediato, por lo que ejecutará durante la creación de una definición de dos puntos para seleccionar este vocabulario en el momento de compilar.

HERE **— addr**

Pone la dirección de la siguiente posición disponible del diccionario.

HEX

Cambia la base de conversión numérica a dieciseis (hexadecimal).

HLD **— addr**

Una variable de usuario que guarda la dirección del último carácter de texto durante la conversión numérica de salida.

HOLD **c —**

Usado entre < # y # > para insertar un carácter ASCII en una cadena numérica de salida, por ejemplo, 2E HOLD pondrá un punto decimal.

I **— n**

Usado dentro de un DO ... LOOP para copiar el índice del bucle en la pila. Vea R.

I/ **— n**

Copia el penúltimo valor de la pila de retorno.

ID. **addr —**

Imprime el nombre de una definición desde la dirección del campo de nombre.

IF **f — (momento de ejecución)**
— addr n (compilar)

Ocurre en una definición de dos puntos en la forma

```
IF(tp) ... ENDIF
IF(tp) ... ELSE(fp) ... ENDIF
```

En el momento de ejecución IF elige la ejecución basada en un banderín booleano. Si f es verdadero (distinto de cero) la ejecución continúa adelante a través de la parte verdadera. Si f está inmediatamente después de ELSE ejecuta la parte falsa. Después de cualquiera de éstas, la ejecución continúa después de ENDIF. ELSE y su parte falsa son opcionales. Si faltan, la ejecución falsa salta hasta inmediatamente después de ENDIF.

Al momento de compilar, IF compila ØBRANCH y reserva espacio para una diferencia en addr. addr y n se usarán después para la resolución de la diferencia y comprobación de errores.

INMEDIATE

Marca la última definición hecha, de tal manera que, al encontrarla al momento de compilar, la ejecutará en vez de compilarla. Por ejemplo, se fija el bit de precedencia en su cabecera. Este método permite a las definiciones manejar situaciones inusuales de compilación, en vez de construirlas en el compilador fundamental. El usuario puede forzar la compilación de una definición INMEDIATE anteponiéndole **COMPILE.**

IN — addr

Una variable de usuario conteniendo la diferencia en bytes, dentro del buffer de texto de entrada vigente (terminal o disco), del que será tomado el siguiente texto. WORD usa y desplaza el valor de IN.

INDEX from to —

Imprime la primera línea de cada pantalla entre los límites from, to. Se usa para ver las líneas de comentario de un área de texto en pantallas de disco.

INIT-DISC

Limpia toda la información del disco-RAM antes de usarla por primera vez.

INTERPRET

El intérprete exterior de texto que ejecuta o compila secuencialmente el texto del canal de entrada (terminal o disco) dependiendo de STATE. Si la palabra no puede ser encontrada después de una búsqueda en CONTEXT y CURRENT, se convierte en un número según la base vigente. Si falla ésto también, aparecerá un mensaje de error seguido de un signo de interrogación. La entrada de texto se tomará según la convención para WORD. Si se encuentra un punto decimal como parte de un número, se dejará un valor de número doble. El punto decimal no tiene otro fin que forzar esta acción. Vea NUMBER.

J **— n**

Se usa dentro de bucles anidados DO. Da el valor del índice del bucle exterior.

KEY **—v**

Pone el valor ASCII de la siguiente tecla del terminal pulsada.

LATEST **— addr**

Pone la dirección del campo del nombre, de la palabra que esté encima del todo en el vocabulario CURRENT.

LEAVE

Fuerza la terminación de un bucle DO ... LOOP en la primera oportunidad, haciendo el límite del bucle igual al valor actual del índice. Este permanece inalterado y la ejecución continúa normalmente hasta que se encuentre LOOP o +LOOP.

LFA **pfa — lfa**

Convierte la dirección del campo de parámetro de una definición del diccionario en la dirección del campo de enlace.

LIMIT **— n**

Una constante que pone la dirección inmediatamente más arriba, de la memoria más alta disponible para un buffer de disco. Normalmente es la memoria más alta del sistema.

LINE **n — addr**

Pone la dirección de la línea n de la pantalla actual. La dirección estará en el área del buffer de disco.

LIST **n —**

Muestra el texto ASCII de la pantalla n en el periférico de salida seleccionado. SCR contiene el número de la pantalla durante este proceso y después.

LIT **— n**

Dentro de una definición de dos puntos, LIT es compilado automáticamente antes de cada número literal de 16 bits encontrado en el texto de entrada. La ejecución posterior de LIT hace que el contenido de la siguiente dirección del diccionario sea colocada (push) en la pila.

LITERAL **n — (compilar)**

Al compilar, toma el valor n de la pila como un literal de 16 bits. Esta definición es inmediata, por lo que se ejecutará durante una definición de dos puntos. El propósito es: xxx [calcular] LITERAL; La compilación se suspende para el cálculo de un valor. Se reanuda la compilación y LITERAL lo compila.

LOAD **n —**

Comienza la interpretación de la pantalla n. La carga terminará al final de la pantalla o con ;S. Vea ;S y -- >.

LOOP **addr n — (compilar)**

Ocurre en una definición de dos puntos en la forma

```
DO ... LOOP
```

En el momento de ejecución, LOOP controla selectivamente la bifurcación hacia atrás, hacia el DO correspondiente, tomando como referencia el índice y el límite. El índice se incrementa en uno y se compara con el límite. El salto atrás, hacia DO, ocurre hasta que el índice iguala o excede al límite, se desechan los parámetros y la ejecución continúa adelante.

Al momento de compilar, LOOP compila (LOOP) y usa addr para calcular el desplazamiento hasta DO. n se usa para verificación de errores.

M* **n1 n2 — d**

Operación matemática de magnitudes mixtas, que halla el producto en doble número con signo, de dos números con signo.

M/ **d n1 — n2 n3**

Un operador matemático de magnitudes mixtas, que pone el resto con signo n2 y el cociente con signo n3, resultado de dividir un dividendo de número doble entre un divisor n1. El resto toma el signo del dividendo.

M/MOD **ud1 u2 — u3 ud4**

Una operación matemática de magnitudes mixtas sin signo que deja un cociente doble en ud4 y el resto en u3, de dividendo doble ud1 y divisor sencillo u2.

MAX **n1 n2 — max**

Pone el mayor de dos números.

MESSAGE **n —**

Imprime en el periférico elegido la línea de texto n relativa a la pantalla 4 de la unidad 0. n puede ser positiva o negativa. MESSAGE puede ser usada para imprimir textos incidentales tales como cabeceras de informes. Si WARNING es cero, el mensaje simplemente se imprimirá como un número. (Sistema de RAM-disco).

MIN **n1 n2 — min**

Pone el menor de dos números.

MINUS **n1 — n2**

Pone el complemento a dos de un número.

MOD **n1 n2 — mod**

Pone el resto de n1/n2, con el mismo signo de n1.

MON

Salida a BASIC o sistema operativo.

NEXT

Este es el intérprete interno que utiliza el puntero IP para ejecutar definiciones FORTH compiladas. No se ejecuta directamente sino que es el puntero de retorno de todos los procesos de código. Actúa poniendo la dirección hacia la que apunta IP en el registro W. Después salta a la dirección señalada por la dirección señalada por W. W señala el campo de código de una definición que contiene, a su vez, la dirección del código que ejecuta esa definición. Este uso de código indirecto entrelazado es uno de los factores de la potencia, portabilidad y extensibilidad de Forth. Las localizaciones de IP y W son específicas del ordenador. (Vea nota anterior sobre convención FORTH).

NFA **pfa — nfa**

Convierte la dirección del campo del parámetro de una definición en su campo de nombre.

NOOP

No operación, en FORTH.

NOT **f — f**

Pone un banderín de falso si en el stack hay un banderín de verdadero, de lo contrario pone un banderín de verdadero. (De hecho ejecuta Ø=).

NUMBER **addr — d**

Convierte una cadena de caracteres que se encuentra en la dirección por una cantidad anterior, a un número doble con signo, usando la base numérica vigente. Si se encuentra un punto decimal en el texto se dará su posición en DPL, sin otro efecto. Si no es posible la conversión numérica, aparecerá un mensaje de error.

OFFSET **— addr**

Una variable de usuario que puede contener un desplazamiento (offset) de bloque para unidades de disco. El contenido de OFFSET se suma al número del stack con BLOCK. Los mensajes de MESSAGE son independientes de OFFSET. Vea BLOCK, MESSAGE. Es REDUNDANTE en FORTH 1.1b

OR **n1 n2 — or**

Pone el valor de los bits un or lógico de dos valores de 16 bits.

OUT **— addr**

Una variable de usuario que contiene un valor incrementado por EMIT. El usuario puede modificar y examinar OUT para controlar el formato de pantalla.

OVER **n1 n2 — n1 n2 n1**

Copia el segundo valor del stack, poniéndolo como primero.

PAD **— addr**

Pone la dirección del buffer de salida de texto, el cual es un desplazamiento (offset) fijo por encima de HERE.

PFA **nfa — pfa**

Convierte la dirección del campo de nombre de una definición compilada a la dirección del campo de parámetro.

PREV **— addr**

Una variable que contiene la dirección del buffer de disco al que más recientemente se ha hecho referencia. El comando UPDATE marca este buffer para que sea posteriormente grabado en disco.

QUERY

Da entrada a 80 caracteres de texto (o hasta que se pulse ENTER) desde el terminal de los operadores. El texto se coloca en la dirección contenida en TIB con IN puesto a cero.

QUIT

Borra el stack de retorno, detiene la compilación y devuelve el control al terminal de los operadores. No aparece mensaje.

R **— n**

Copia el primer valor del stack de retorno en el stack de cálculo.

R # **— addr**

Una variable de usuario que puede contener la localización de un cursor de edición, u otra función relacionada con el fichero.

R/W **addr blk f —**

El enlace estándar de fig-FORTH para grabar o cargar de disco. addr especifica el buffer de bloque de origen o destino. blk es el número secuencial del bloque de referencia; y f es un banderín, f=0 para grabar, y f=1 para cargar. R/W, determina la localización en almacenamiento masivo, realiza la lectura y grabación y comprueba los errores.

R > **— n**

Quita el primer valor del stack de retorno y lo coloca en el stack de cálculo. Vea >R y R.

RØ — addr

Una variable de usuario conteniendo la localización inicial del stack de retorno. Se llama 'R-cero'. Vea RP!.

REPEAT addr n — (al compilar)

Usado en una definición de dos puntos en la forma

```
BEGIN ... WHILE ... REPEAT
```

Al momento de ejecutar, REPEAT fuerza una bifurcación incondicional hacia atrás, hasta justo después del correspondiente BEGIN.

Al momento de compilar, REPEAT compila BRANCH y el desplazamiento (offset) desde HERE a addr. n se utiliza para comprobar errores.

ROT n1 n2 n3 — n2 n3 n1

Rota los tres primeros valores del stack, colocando el tercero como primero.

RP@ — addr

Pone el valor vigente en el registro del puntero del stack de retorno.

RP!

Un procedimiento dependiente del ordenador para inicializar el puntero del stack de retorno desde la variable de usuario RØ.

S->D n — d

Extiende un número sencillo en número doble.

SØ — addr

Una variable de usuario que contiene el valor inicial del puntero del stack. Se llama 'S-cero'. Vea SP!

SCR **— addr**

Una variable de usuario que contiene el número de la pantalla más recientemente mencionada por LIST.

SIGN **n d — d**

Guarda un signo ASCII delante de una cadena numérica convertida de salida en el siguiente buffer de salida, cuando n es negativa. n se desecha, pero el número doble d se conserva. Debe ser usado entre < # y # >.

SMUDGE

Usada durante la definición de palabras para cambiar el bit de 'smudge' en el campo de nombre de una definición. Impide que se encuentre una definición incompleta cuando se busca en el diccionario, hasta que la compilación se termina sin errores.

SP!

Un procedimiento dependiente del ordenador para inicializar el puntero del stack desde SØ.

SP@ **— addr**

Un procedimiento dependiente del ordenador para devolver la dirección de la posición del stack al primer lugar de stack, como estaba antes de que SP@ fuese ejecutado. (Por ejemplo, 1 2 SP@ @ ...escribiría 2 2 1).

SPACE

Envia un espacio en blanco ASCII al periférico de salida.

SPACES **n —**

Envia n espacios en blanco ASCII al periférico de salida.

STATE **— n**

Una variable de usuario que contiene el estado de la compilación. Un valor distinto de cero indica compilación.

SWAP **n1 n2 — n2 n1**

Intercambia los dos primeros valores en el stack.

TASK

Una palabra no ejecutable que puede delimitar las aplicaciones. Al indicar que 'olvide' (FORGET) a TASK y re-compilar, se puede desechar una aplicación en su totalidad.

TEXT **c —**

Acepta el texto que sigue en PAD. c delimita el texto.

THEN

Otro nombre de ENDIF.

TIB **— addr**

Una variable del usuario que contiene la dirección del buffer del terminal de entrada.

TOGGLE **addr b —**

Complementa el contenido de addr con la disposición del bit b.

TRAVERSE **addr1 n — addr2**

Desplaza en toda la longitud de un campo de nombre de una variable fig-FORTH. add1 es la dirección del byte de longitud o de la última letra. Si n=1 se deplaza hacia la parte alta de la memoria; si n=-1 se desplaza hacia la parte baja. La addr resultante es la del otro extremo del nombre.

TRIAD **scr —**

Muestra en el periférico elegido las tres pantallas, incluyendo a la numerada scr, comenzando con una divisible entre tres. La salida es apropiada para registros fuente de textos.

TYPE **addr count —**

Transmite caracteres del contador de addr al periférico de salida elegido.

U< **u1 u2 — f**

Pone el valor booleano de una comparación menor-que sin signo. Pone f=1 si u1<u2; de lo contrario pone 0. Esta función debe usarse al comparar direcciones de memoria.

U* **u1 u2 — ud**

Pone el producto sin signo en número doble de dos números sin signo.

U. **u —**

Imprime un número de 16 bits sin signo convertido según la BASE. Le sigue un espacio en blanco.

U.R **u n —**

Imprime el número sin signo u alineado a la derecha en un campo cuya anchura es n. No se imprime ningún espacio en blanco detrás.

U/MOD **ud u1 — u2 u3**

Pone el resto sin signo u2 y el cociente sin signo u3, resultado del dividendo doble sin signo ud y el divisor sin signo u1.

UNTIL **f — (al momento de ejecución)**

addr n — (al compilar)

Ocurre con una definición de dos puntos de la forma:

```
BEGIN ... UNTIL
```

Al momento de ejecución, UNTIL controla la bifurcación condicional hacia atrás al correspondiente BEGIN. Si f es falso la ejecución regresa hasta justo después de BEGIN; si es verdadero la ejecución continua adelante.

Al momento de ejecutar, UNTIL compila ØBRANCH y un desplazamiento (offset) desde HERE hasta addr. n se utiliza para la comprobación de errores.

UPDATE

Marca el bloque al que más recientemente se ha hecho referencia (señalado por PREV) como modificado. El bloque será posteriormente transferido automáticamente a disco, si se necesitara su buffer para guardar un bloque diferente.

USE — addr

Una variable conteniendo la dirección del buffer de bloque que será usado el siguiente como el menos recientemente escrito.

USER n —

Una palabra definible usada en la forma:

```
n VARIABLE cccc
```

El campo de parámetro de cccc contiene n como desplazamiento fijo relativo al registro del puntero de usuario UP para esa variable. Al ejecutar cccc posteriormente, coloca en el stack como dirección de almacenamiento de esa variable en particular, la suma de su desplazamiento más la dirección de base del área del usuario.

VARIABLE

Una palabra de definición usada en la forma:

```
n USER cccc
```

Al ejecutar VARIABLE crea la definición cccc con su campo de parámetro inicializado a n. Al ejecutar cccc, la dirección de su campo de parámetro (conteniendo n) se pone en el stack, para poder acceder a ella para extraer o guardar.

VLIST

Lista los nombres de las definiciones en el vocabulario de contexto. ESC terminará el listado.

VOC-LINK — addr

Una variable de usuario conteniendo la dirección de un campo en la definición del vocabulario creado más recientemente. Todos los nombres de vocabulario están enlazados por estos campos para permitir el control, al poder 'olvidar' (FORGET) múltiples vocabularios en toda su extensión.

VOCABULARY

Una palabra de definición usada en la forma:

```
VOCABULARY cccc
```

para crear una definición de vocabulario cccc. El uso subsiguiente de cccc lo convertirá en el vocabulario CONTEXT, en el cual primero busca INTERPRET. La secuencia cccc DEFINITIONS también convertirá a cccc en el vocabulario CURRENT, donde se colocan nuevas definiciones.

En fig-FORTH, cccc estará de tal forma relacionada que incluirá todas las definiciones del vocabulario en la que cccc misma es definida. Todo el vocabulario estará, por último, unido a Forth. Por convención los nombres de vocabulario se declarán IMMEDIATE. Vea VOC-LINK.

WARNING — addr

Una variable de usuario conteniendo un valor que controla los mensajes. Si =1 el disco está presente y la pantalla 4 de la unidad 0 es la localización de base para los mensajes. Si =0, no hay disco presente y los mensajes serán mostrados por números. Si =-1, ejecuta (ABORT) para procesos definidos por el usuario. Vea MESSAGE, ERROR.

WHILE f — (al ejecutar)

ad1 n1 — ad1 n1 ad2 n2

Ocurre en una definición de la forma:

```
BEGIN ... WHILE(tp) ... REPEAT
```

Al momento de ejecutar, WHILE elige la ejecución condicional basado en un banderín booleano f. Si f es verdadero (distinto de cero), WHILE continúa la ejecución de la parte verdadera hasta REPEAT, que bifurca hacia atrás hasta BEGIN. Si f es falso (cero), la ejecución salta hasta después de REPEAT, saliendo de la estructura.

Al momento de compilar, WHILE emplaza (ØBRANCH) y pone ad2 del desplazamiento (offset) reservado. Los valores del stack son resueltos por REPAT.

WIDTH — addr

En fig-FORTH es una variable de usuario conteniendo el máximo número de letras guardadas en la compilación de un nombre de definición. Debe ser de 1 a 31, con un valor por defecto de 31. Los caracteres y la cantidad son guardados, hasta el valor en WIDTH. El valor puede ser modificado en cualquier momento dentro de los límites mencionados.

WORD c —

Lee los siguientes caracteres de texto del canal de entrada que es interpretado, hasta que encuentra un delimitador c, guardando la cadena de caracteres comenzando en el buffer del diccionario HERE. WORD pone el contador de caracteres en el primer byte, los caracteres y finaliza con dos o más espacios en blanco. Las primeras ocurrencias de c son ignoradas. Si BLK es cero, el texto se toma del buffer de entrada del terminal, de lo contrario se toma del bloque de disco guardado en BLK. Vea BLK, IN.

X

Este es pseudónimo de 'nulo' o acepción del diccionario para un nombre de un sólo carácter nulo ASCII. Es el procedimiento de ejecución para terminar la interpretación de una línea de texto del terminal o de un buffer de disco, ya que ambos buffers siempre tiene un nulo al final.

XOR n1 n2 — xor

Pone el valor exclusivo-or lógico de dos valores.

[

Usado en una definición de dos puntos en la forma:

```
: xxx [palabras] MORE ;
```

Suspende la compilación. Las palabras que siguen a [son ejecutadas, no compiladas. Esto permite calcular o hacer excepciones de compilación antes de reanudar la compilación con] . Vea LITERAL,] .

COMPILE

Usado en una definición de la forma:

```
: xxx [COMPILE] FORTH ;
```

[COMPILE] forzará la compilación de una definición inmediata, que de otra forma se ejecutaría durante la compilación. El ejemplo anterior seleccionará el vocabulario FORTH al ejecutar xxx, y no cuando compile.

]

Reanuda la compilación, hasta la terminación de una definición de dos puntos. Vea [.